НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
"КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО"
ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ

До захисту допущено:
В.о.завідувача кафедри
_____ Оксана ТИМОЩУК
«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системний аналіз»

спеціальності 124 «Системний аналіз і управління»

на тему: «СИСТЕМА ПЕРЕНЕСЕННЯ СТИЛЮ ЗОБРАЖЕНЬ ЗА ДОПОМОГОЮ ШТУЧНИХ ГЛИБИННИХ НЕЙРОННИХ МЕРЕЖ**»**

Виконав:
студент IV курсу, групи КА-61
Таранов Я.А. _____

Керівник:
доцент, к.т.н. Дідковська М.В _____

Консультант з економічного розділу:
доцент к. е. н. Шевчук О. А. _____

Консультант з нормоконтролю:
доцент, к. т. н. Коваленко А. Є. _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 124 «Системний аналіз»

Освітньо-професійна програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
В.о.завідувача кафедри
_____ Оксана ТИМОЩУК
«___»_____20__ р.

ЗАВДАННЯ
на дипломну роботу студенту
Таранов Ярослав Андрійович

1. Тема роботи «Система перенесення стилю зображень за допомогою штучних глибинних нейронних мереж», керівник роботи Дідковська Марина Віталіївна, доцент, к. т. н., затверджені наказом по університету від «_25_»_травня_ 20_20_ р. №_1143-с__

2. Термін подання студентом роботи 08 травня 2020 року _____

3. Вихідні дані до роботи

4. Зміст роботи

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

6. Консультанти розділів роботи*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
4	доцент к. е. н., Шевчук О. А.		29.05.2020

7. Дата видачі завдання _____

* Якщо визначені консультанти. Консультантом не може бути зазначено керівника дипломної роботи.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Отримане завдання	13.04.2020	Виконано
2	Збір інформації	20.04.2020	Виконано
3	Ознайомлення з літературою	27.04.2020	Виконано
4	Аналіз вимог завдання і вибір засобів розв’язання поставленної задачі	04.05.202	Виконано
5	Розробка програмного продукту	11.05.2020	Виконано
6	Виконання функціонально-вартісного аналізу	29.05.2020	Виконано
7	Отримання висновків після тестування	30.05.2020	Виконано
8	Оформлення дипломної роботи	08.06.2020	Виконано
9	Оформлення допуску до захисту та подача роботи до ДЕК	12.06.2020	Виконано

Студент Ярослав Таранов

Керівник Марина Дідковська

РЕФЕРАТ

Дипломна робота: 86 с., 6 табл., 15 рис., 2 додатка, 20 джерел.

ПЕРЕНЕСЕННЯ СТИЛЮ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ.

Об’єкт дослідження: система перенесення стилю, що дозволяє змістити стиль одного зображення до стилю іншого, не втрачаючи вихідний зміст.

Предмет дослідження: методи перенесення стилю, що базуються на згорткових нейронних мережах

Мета дослідження: створити інструмент контрольованого перенесення стилю, що дозволяє дослідницькі експерименти та інші застосування

Постановка задачі: розробити систему перенесення стилю на базі математичного апарату згорткових нейронних мереж

Програмний продукт було розроблено з використанням мови програмування Python, графічна складова була написана за допомогою програмною бібліотеки PyQt.

ABSTRACT

Thesis: 86 p., 6 tabl., 15 fig., 2 appends., 20 sources.

STYLE TRANSER, CONVOLUTIONAL NEURAL NETWORKS.

Object of research: style transfer system that allows to shift style of an image to style of another one while preserving content

Subject of research: convolutional neural network-based algorithms for style transfer

The purpose of the study: to create tool for controllable style transfer, allowing both research experiments and custom usage

Problem statement: to develop a style transfer system on the basis of the mathematical apparatus of convolutional neural networks

The software product was developed using Python programming language, with GUI part based on PyQt library.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ШНМ – Штучні нейронні мережі

ЗНМ — Згорткові нейронні мережі

ВСТУП

Важливою особливістю, притаманною людському розуму, є здатність до творчості. Однією з рис цієї властивості є застосування певного стилю чи текстури до спостереженого зображення. Хоча загалом можна розглядати цей процес для будь-якої складової сприйняття, у тому числі звук чи сприйняття тексту, найбільш яскраво ця особливість проявляється саме для зображень.

Як і для більшості подібних задач, не існує строгого математичного апарату, який описує те, як цю задачу розв’язує людський розум. Водночас, клас математичних моделей, що об’єднується під назвою згорткових нейронних мереж, має певні спільні риси з реальними біологічними системами візуального сприйняття.

Розв’язання задачі перенесення стилю з достатньою якістю, не втрачаючи семантичний зміст вхідного зображення, можна вважати як кінцевою метою, що дозволить об’єднувати різноманітні зображення в єдині для сприйняття поєднання, так і кроком на шляху до розуміння візуального сприйняття та творчого процесу людини.

Дипломна робота складається з чотирьох розділів.

Перший розділ оцінює актуальність поставленої задачі, виконує огляд існуючих для її вирішення алгоритмів та оцінює їх якість та оптимальність.

Другий розділ описує теоретичні основи математичного апарату згорткових нейронних мереж.

Третій розділ описує розробку програмного продукту і побудований алгоритм, що будується на згорткових мережах.

Четвертий розділ виконує функціонально-вартісний аналізу отриманого програмного продукту.

РОЗДІЛ 1 АНАЛІЗ ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ МЕТОДІВ ПЕРЕНЕСЕННЯ СТИЛЮ

1.1 Задача перенесення стилю

У образотворчому мистецтві, особливо живописі, люди оволоділи майстерністю створювати неповторні візуальні переживання через складання складної взаємодії між змістом та стилем зображення. Поки що алгоритмічна основа цього процесу є невідомою і не існує штучної системи, що могла б відтворити цей процес. Однак в інших ключових сферах візуального сприйняття, таких як розпізнавання предметів і обличь, були побудовані моделі на основі глибинних мереж, що досягають та в деяких завданнях перевищують показники, досягнуті людиною.

Перенесення стилю з одного зображення на інше можна вважати проблемою перенесення текстури. Постановка задачі полягає у синтезі зображення з текстурою, відповідною зображенню стилю, накладаючи обмеження на збереження змісту – семантичного навантаження вхідного зображення.

Найперше застосування, що спадає на думку у контексті перенесення стилю – це імітація або створення робіт мистецтва, дизайн та споріднені напрями. Достатньо частою задачею у цих областях є створення цілісного образу і враження від об’єкту, що складається з багатьох різнотипних елементів. Проте варто зауважити, що цілком можливими і доречними можуть бути й інші застосування цієї методики, особливо в тих областях, де необхідно отримати цілісне для сприйняття людиною зображення. Як приклад, можна запропонувати застосування перенесення стилю до супутникових знімків поверхні Землі чи інших планет, для

отримання цілісної картини, що компенсує різні зовнішні та внутрішні умови, у яких знаходився супутник, і що значно впливають на візуальне представлення зображення. В цьому контексті важливим є максимально точне збереження семантичної інформації щодо розташування видимих об’єктів, рельєфу тощо, при цьому збереження загального оригінального вигляду знімку не є настільки критичним.

1.2 Класичні алгоритми перенесення стилю

Для синтезу текстур існує великий діапазон потужних непараметричних алгоритмів, які можуть синтезувати фотореалістичні природні текстури шляхом перекомпонування пікселів заданої текстури джерела.

Більшість попередніх алгоритмів передачі текстури покладаються на ці непараметричні методи синтезу текстури, використовуючи різні способи збереження і відтворення структури цільового зображення.

Хоча ці алгоритми досягають чудових результатів, всі вони мають однакове фундаментального обмеження: їх дія базується лише на взаємному розташуванні пікселів зображення, однак не має можливості оцінити високорівневу семантичну інформацію, яку сприймає людина, таку як наявність, сутність і можливий взаємозв’язок об’єктів на зображенні. Однак для оптимальної роботи алгоритм перенесення стилів повинен мати можливість отримувати саме змістовну складову вхідного зображення, а потім передавати процедурі передачі текстури його семантичний зміст для перенесення на його стилізовану версію.

Тому метою є створення системи перетворень зображень, які незалежно моделюють варіації змісту семантичної складової зображення та його стилю. Такі

факторизовані представлення раніше були досягнуті лише для визначеного набору підмножин природних зображень, таких як обличчя в умовах різного освітлення, символи, що відповідають різним шрифтам, рукописні цифри та номери будинків.

1.3 Застосування глибинних нейронних мереж

За останні 10 років найефективніші системи штучного інтелекту - такі як розпізнавачі мови на смартфонах чи останній автоматичний перекладач Google - були створені за допомогою напрямку, що називається глибинне навчання.

Глибинне навчання насправді є новою назвою для підходу до штучного інтелекту під назвою нейронні мережі, які то починали використовуватись, то виходили з активного застосунку протягом останніх 70 років. Нейронні мережі були головним напрямком досліджень як в галузі нейрології, так і в галузі інформатики до 1969 року, поки недоцільність на той час їх використання не була доведена Марвіном Мінським та Сеймуром Пертом. Хоча частково ця методика відродилась 1980-х роках, її піднесення тривало порівняно недовго і припинилось вже на початку 2000-х років. Але після 10-річної паузи нейронні мережі стрімко перейшли до етапу їх найактивнішого на цей момент розвитку, що триває і на цей момент.

1.3.1 Обробка згортковими мережами візуальної інформації

У сфері зображень і візуальної інформації, протягом останніх десятиріч були створені потужні комп'ютерні системи зору, які вчаться витягувати

семантичну інформацію високого рівня з природних зображень. Було показано, що згорткові нейронні досягають надзвичайно високих показників у ряді завдань, таких як розпізнавання об'єктів, вчаться отримувати високорівневий семантичний вміст зображень в загальних представленнях функцій, та здатні узагальнювати його між різними наборами даних та завданнями обробки візуальної інформації, включаючи розпізнавання текстур та класифікацію художнього стилю

Багато класичних проблем можна охарактеризувати як завдання з перетворення зображення, де система отримує певне вхідне зображення і за певним алгоритмом отримує нове. Приклади обробки зображень включають в себе локалізацію об’єктів, надроздільну здатність та колоризацію, де вхід є погіршеним зображенням (зашумленим, з низькою роздільною здатністю або відтінками сірого), а мета – отримати якісне кольорове зображення. Приклади задач комп’ютерного зору включають семантичну сегментацію та оцінку глибини, де вхідним сигналом є кольорове зображення та вихідне зображення кодує семантичну або геометричну інформацію про сцену.

Клас глибинних нейронних мереж, які є найпотужнішими серед існуючих різновидів в задачах обробки зображень, називається згортковими нейронними мережами. Згорткові нейронні мережі складаються з згорткових шарів – обчислювальних одиниць, що ієрархічно обробляють візуальну інформацію в режимі подачі. Кожен згортковий шар можна представити як сукупність фільтрів, кожен з яких спостерігає і кількісно оцінює певну числову характеристику зображення. Таким чином, вихід кожного шару складається з так званих активаційних карт, перетворених версій зображення. Окремі дослідження спрямовані на виявлення конкретних ознак зображення, які у неявному вигляді зберігаються у активаційних картах, і саме факт, що ця інформація є часто високорівневою та аналогічному тій, яку отримує із зображення людина, надав можливість будувати ряд алгоритмів, базуючись на їх значеннях.

1.4 Існуючі методи застосування згорткових мереж до перенесення стилю

Один із підходів до вирішення завдань перетворення зображення полягає у тому, щоб навчити згорткову нейронну мережу генерувати стилізовані зображення за допомогою тренування зі вчителем, використовуючи як метрику відстань між відповідними пікселями.

Однак попарні відстані між пікселями зображення, використані цими методами, не охоплюють сприйняття відмінності між вхідними та створеними зображеннями. Наприклад, розглянемо два однакові зображення, зміщені один від одного на один піксель: незважаючи на їх подібність в очах людини, будь-яка метрика дала б високе значення відстані, якщо її застосувати до попарного порівняння пікселів. Інші роботи показали, що можна створювати високоякісні зображення використовуючи функції сприйняття втрат, засновані не на відмінностях пікселів, а на відмінностях між витягнутими представленнями зображень високого рівня з попередньо перевірених згорткових нейронних мереж. Зображення створюються за рахунок мінімізації функції втрат. Ця стратегія була застосована до інверсії функцій, а також до синтезу текстур та передачі стилю. Ці підходи створюють високоякісні зображення, але повільні, оскільки вимагає висновку вирішення проблеми оптимізації. Також вони вимагають наявність даних для навчання, які в контексті перенесення стилю фактично неможливо зібрати. Як наслідок, однією з головних задач є побудова методу, що не лише справляється із задачею перенесення стилю, а і здатний виконати її з обмеженими ресурсами та без прикладів бажаного результату дії алгоритму.

Висновки до розділу 1

Перенесення стилю – задача, яку, незважаючи на інтуїтивно просте розуміння людиною, складно задати алгоритмічно. Хоча існує ряд методів, які в тій чи іншій мірі вирішують цю проблему, вони в абсолютній більшості є обмеженими або відносно примітивною структурою, що не зберігає семантичні дані, або ресурсозатратністю алгоритму та необхідністю виконання япроцесу навчання. Останні часом найбільш перспективні методи застосовують згорткові глибинні нейронні мережі для розв’язку цієї проблеми.

РОЗДІЛ 2 ТЕОРЕТИЧНІ ОСНОВИ ТА ЕВОЛЮЦІЯ ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ

2.1 Штучні нейронні мережі

Штучні нейронні мережі – це клас алгоритмів, у загальному натхненний біологічними нейронними мережами, якими є мозок людей і тварин. Класичні штучні нейронні системи "вчаться" виконувати завдання, отримуючи приклади бажаного виконання. Нейронні мережі здатні узагальнювати і з високою точністю наближувати широкий клас функцій, ніж більшість класичних методів. А метод зворотного поширення помилки та споріднені йому дозволяють знайти локальний чи глобальний оптимум в латентному просторі параметрів, який часто є занадто великий для оптимізації класичними методами. Завдяки таким властивостям штучні нейронні мережі знайшли широке застосування в різноманітних прикладних областях, при чому одна й та сама чи споріднені архітектури мережі часто досягають високих результатів на неспоріднених між собою задачах і не отримуючи жодної додаткової інформації щодо предметної області.

Штучні нейронні мережі складаються з сукупності з'єднаних одиниць або вузлів, які називають штучними нейронами, що наближено моделюють нейрони в біологічному мозку. Кожне з'єднання, як і синапси в біологічному мозку, може передавати сигнал певній групі нейронів. Штучний нейрон, який сприймає сигнал, виконує перетворення на ньому та передає наступній групі.

У штучних нейронних мережах "сигнал", що передають штучні нейрони, є дійсним числом, а вихід кожного нейрона обчислюється деякою нелінійною функцією суми його входів. З'єднання називаються ребрами. Нейрони та ребра

зазвичай мають вагу, яка коригується у міру тривалості навчання. Вага збільшує або зменшує силу сигналу при з'єднанні. Нейрони можуть мати порогове значення, внаслідок чого сигнал надсилається лише в тому випадку, якщо сукупний сигнал перетинає цей поріг. Зазвичай нейрони агрегуються в групи, що називають шарами. Різні шари можуть виконувати різні перетворення на отриманих сигналах.

Початкова мета підходу штучних нейронних мереж полягала у тому, щоб змоделювати розв'язання проблем людським мозком. Проте поступово увага змістилася до виконання конкретних завдань, що призводить до відхилень від біології. Штучні нейронні мережі використовуються у різних сферах, зокрема розпізнаванні та класифікації візуальних образів, розпізнаванні мови, машинному перекладі, настільних та відеоіграх, медичній діагностиці та навіть для створення витворів мистецтва.

2.1.1 Одношаровий перцептрон

Найпростішим прикладом нейронної мережі є одношаровий перцептрон. Його функціональне представлення є досить простим:

$$f(x) = \begin{cases} 1, & \text{якщо } wx + b > 0 \\ 0, & \text{якщо } wx + b \leq 0 \end{cases} \tag{2.1.1}$$

Інакше кажучи, дію перцептрона можна розглядати як композицію лінійного перетворення та функції Хевісайда. Його можна застосувати в найпростіших задачах бінарної класифікації, де вхідні дані у векторному форматі можуть бути лінійно розподілені за двома класами з достатньо високою точністю. Хоча

одношаровий перцептрон має обмежене прикладне застосування, саме його модифікації є основною атомарною складовою більшості сучасних штучних нейронних мереж.

2.1.2 Багатошаровий перцептрон

Багатошаровий перцептрон є більш складною і практичною штучною нейронною мережею за одношаровий. Його назва спричинена тим, що такі мережі складаються з послідовно з’єднаних модифікованих одношарових перцептронів. Багатошаровий перцептрон складається з щонайменше трьох шарів вузлів: вхідного шару, прихованого шару та вихідного шару. За винятком вхідних вузлів, кожен вузол є нейроном, який використовує функцію нелінійної активації. Багатошаровий перцептрон використовує типове для нейронних мереж навчання зі вчителем за допомогою методу зворотного поширення помилки.

2.2 Згорткові нейронні мережі

Структура згорткових нейронних мереж також черпає натхнення у схемі зв'язку нейронів у людському мозку, а саме є наближеним моделюванням зорової кори. Окремі нейрони реагують на подразники лише в обмеженій області зорового поля, яку називають рецептивним полем. Такі нейрони групуються у структури, що

за допомогою частково перекриваючихся рецептивних полів мають змогу проаналізувати всю зону спостереження (вхідне зображення).

Згорткові мережі здатні успішно фіксувати просторові та часові залежності в зображенні за допомогою застосування відповідних фільтрів. Архітектура забезпечує краще пристосування до набору даних зображення за рахунок зменшення кількості задіяних параметрів та повторного використання ваг. Структурною одиницею згорткової нейронної мережі є згорткоий нейронний шар. Варто помітити, що незважаючи на назву, дія згорткового шару не є функцією згортки, хоча і має схожий математичний запис.

2.2.1 Згортковий нейронний шар

Параметри згорткового нейронного шару складаються з набору фільтрів, що вивчаються за допомогою навчання зі вчителем. Історично першими були мережі, де фільтри задавались вручну і були спрямовані на виявлення конкретних особливостей зображення. Однак сучасні імплементації згорткових мереж в абсолютній більшості використовують саме навчаємі фільтри, оскільки вони є набагато більш потужними у плані множини наближуємих функцій.

Кожен фільтр є просторово обмежений (по ширині та висоті), але поширюється на всі канали вхідного зображення. Наприклад, типовий фільтр першого шару згорткової мережі може мати розмір 5x5x3 (тобто ширина та висота пікселів 5 пікселів і 3, оскільки зображення мають глибину 3, кольорові канали). Під час проходу вперед ми просуваємо (точніше, обертаємо) кожен фільтр по ширині та висоті вхідного об'єму та обчислюємо крапки дотику між записами фільтра та входом у будь-якому положенні. Коли ми просунемо фільтр по ширині

та висоті вхідного об'єму, ми створимо двовимірну карту активації, яка дає відповіді цього фільтра у кожному просторовому положенні.

Інтуїтивно зрозуміло, що мережа засвоїть фільтри, які активуються, коли вони побачать певний тип візуальної функції, наприклад, край певної орієнтації або пляма якогось кольору на першому шарі, або, зрештою, цілі сотові або колесоподібні візерунки на вищих шарах мережі . Тепер у нас буде цілий набір фільтрів у кожному згортковому шарі (наприклад, 12 фільтрів), і кожен з них створить окрему двовимірну карту активації. Ми будемо розташовувати ці карти активації по глибинному виміру і виробляти вихідний об'єм.

2.2.2 Активаційні карти

Коли згорткові нейронні мережі навчаються розпізнаванню об'єктів, в латентному просторі їх активацій виникають визначені представлення зображення, що складають певну ієрархію його обробки.

Ітеруючись скрізь шари, такі представлення поступово віддаляються до таких, що містять низькорівневу піксельну інформацію до високорівневої семантичної інформацію. Ми можемо безпосередньо візуалізувати інформацію, що спостерігається кожним фільтром визначеного шару містить інформацію про вхідне зображення шляхом реконструкції зображення лише з карт функцій у цьому шар. Вищі шари в мережі захоплюють вміст високого рівня з точки зору об'єктів та їх розташування у вхідному зображенні, але не обмежують точні значення пікселів реконструкції. Тоді як з активаційних мап нижніх шарів просто відтворити точні значення пікселів вихідного зображення.

Метою цієї роботи є створення системи на основі глибинної штучної нейронної мережі, що створює художні образи високої перцептивної якості. Система використовує представлення зображень в латентному просторі активацій нейронної мережі для розділення та рекомбінації змісту та стилю довільних зображень, надаючи алгоритм створення нових художніх образів. Крім того, у світлі вражаючої подібності між штучно згенерованими та створеними людиною зображеннями, це надає можливість хоча б наближено зрозуміти алгоритмі того, як люди створюють та сприймають художні образи.

Активаційні карти, засвоєні високоефективними згортковими нейронними мережами, можуть використовуватися для самостійної обробки та маніпулювання змістом та стилем природних зображень. Ми представляємо нейронний алгоритм художнього стилю, новий алгоритм для передачі стилю зображення. Концептуально це алгоритм передачі текстури, який обмежує метод синтезу текстур за допомогою представлення особливостей сучасних згорткових нейронних мереж. Оскільки текстурна модель також заснована на глибоких уявленнях зображень, метод передачі стилів елегантно зводиться до проблеми оптимізації в межах однієї нейронної мережі. Нові зображення генеруються шляхом пошуку попереднього зображення для відповідності представленим зображенням прикладних зображень. Цей загальний підхід раніше застосовувався в контексті синтезу текстур та для поліпшення розуміння глибоких уявлень про зображення

Для отримання уявлення про стиль вхідного зображення ми спочатку використовуємо простір функцій для збору текстурної інформації. Цей простір побудований на основі активаційних карт в кожному шарі мережі. Він складається з кореляцій між різними відгуками фільтрів над просторовим розмахом карт функцій. Обчислюючи функцію кореляції декількох шарів, ми отримуємо стаціонарне, багатомасштабне подання вхідних даних зображення, яке фіксує свою текстурну інформацію, але не глобальну композицію.

У роботах, що досліджували особливості того, які саме елементи зображення сприймаються активаційними картами було візуалізовано інформацію, захоплену цими латентними просторами різних шарів мережі. Як наслідок, були отримані версії вхідного зображення, які фіксують його загальний вигляд у плані певних локалізованих структур. Причому розмір та складність локального образу структури з вхідного зображення збільшуються вздовж шарів згорткової мережі, результат, який можна пояснити за рахунок збільшення розмірів сприйнятливих полів та складності функції.

Завдяки розподіленню активаційних карт на ті, що передають низькорівневу інформацію щодо форм на зображенні, і ті, що мають високорівневе семантичне навантаження, можемо зробити висновок, що уявлення про зміст та стиль у згортковій нейронній мережі є відокремленими. Звідси випливає, що можливо маніпулювати обома складовими незалежно, щоб створити нові, сприйняті значущі зображення.

2.3 Повторне застосування навчених нейронних мереж

Традиційно алгоритми машинного навчання та глибокого навчання розроблені для ізольованої роботи, тобто модель навчається розв'язувати одну певну задачу. Моделі повинні бути відновлені з нуля, коли зміниться розподіл даних або вхідний простір. Трансферне навчання - це ідея подолання ізольованої парадигми навчання та використання знань, отриманих для одного завдання, для вирішення суміжних.

Ключовою мотивацією до застосування цього методу, особливо у контексті глибокого навчання, є той факт, що для більшості моделей, які вирішують складні

проблеми, потрібна велика кількість даних, і отримання величезної кількості даних, що мають повний опис, для моделей, що навчаються зі вчителем, може бути дуже важким, враховуючи час та зусилля потрібно позначити точки даних. Простим прикладом може бути створений Стенфордським університетом набір даних ImageNet, який містить мільйони зображень, що відносяться до різних категорій.

2.4 Архітектури згорткових нейронних мереж

Історично, розвиток нейронних мереж просувався завдяки створенням окремими академічними та комерційними структурами принципово нових архітектур, структура яких подалі застосовуються, з одного боку, як база для побудови нових мереж, з іншого – застосування самих моделей за допомогою трансферного навчання на нових даних.

2.4.1 AlexNet

Ця архітектура була однією з перших глибоких мереж, що підштовхнула точність класифікації зображень з ImageNet і є суттєвим кроком порівняно з традиційними методологіями. Вона складається з 5 згорткових шарів з наступними 3 повністю з'єднаними шарами (рис. 2.1). Згорткові шари при цьому поєднані так званими операціями «пулінгу», що виражаються взяттям максимуму активаційної карти по кожному каналу та по кожній області заданого розміру.

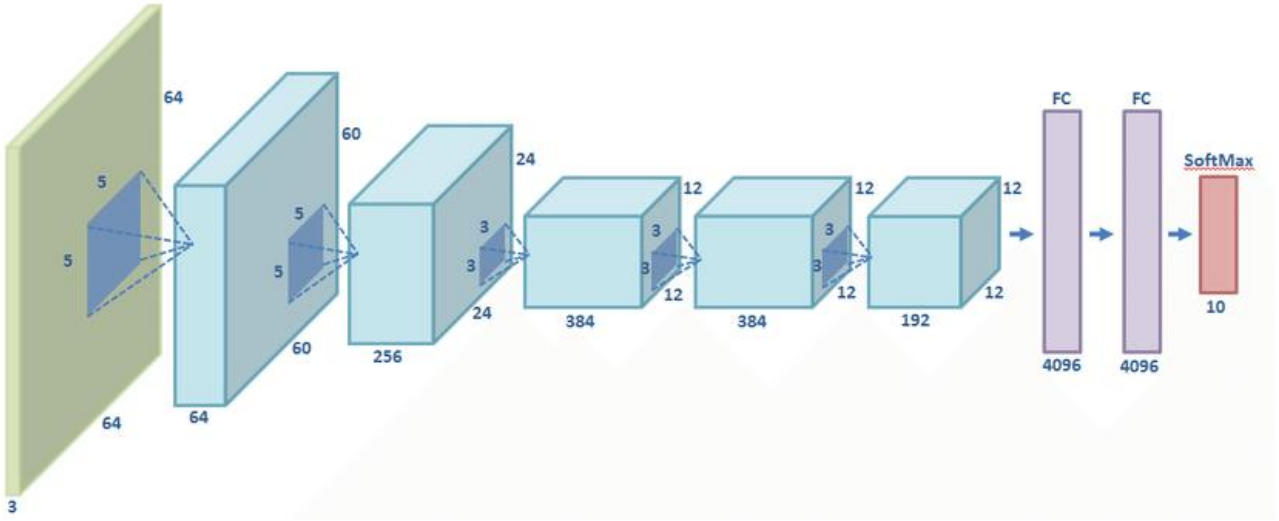

Рисунок 2.1 – Структура згорткової нейронної мережі AlexNet

Запропонована Алексом Крижевським мережа використовує ReLu (випрямлений лінійний блок) для нелінійної частини, а не функцію гіперболічного тангенсу або сигмоїду, що було раніше стандартом для традиційних нейронних мереж.

ReLu математично представляється таким чином:

$$f(x) = max(0, x) \tag{2.41}$$

На рисунку 2.1 можна побачити порівняння графіку ReLu з класичними активаційними функціями.

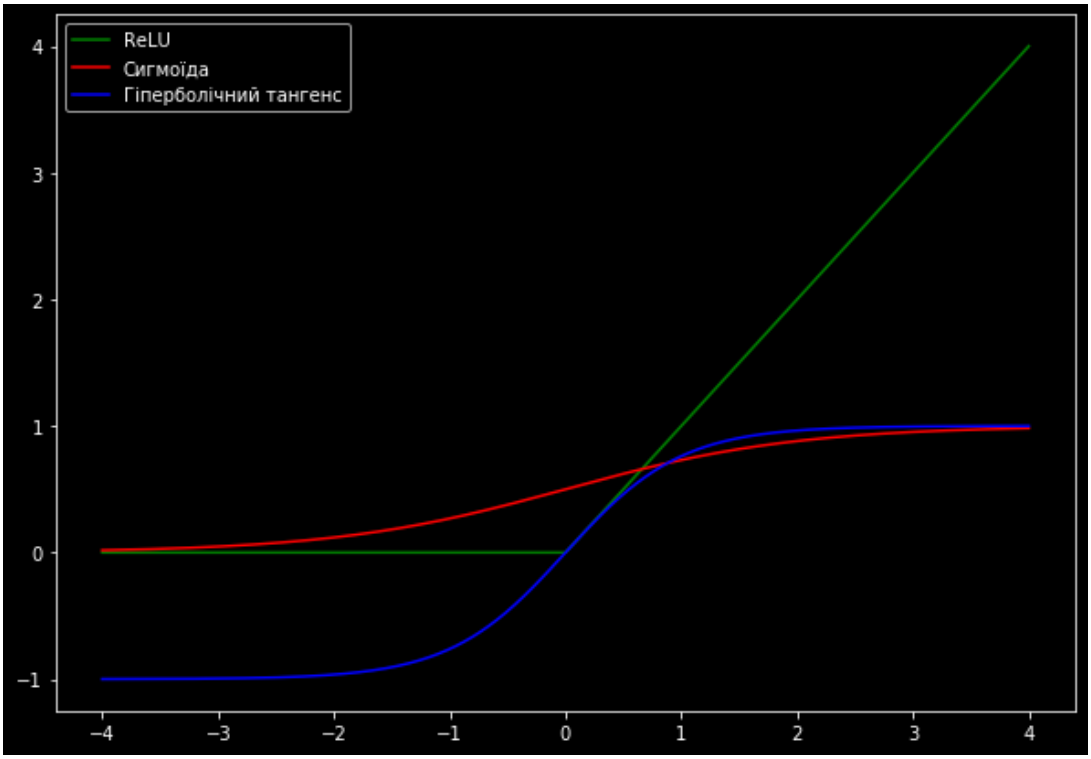

Рисунок 2.1 – Графіки активаційних функцій

Перевага ReLu перед сигмоїдою полягає в тому, що він тренується набагато швидше, оскільки похідна сигмоїди стає дуже малою в області насичення, і тому оновлення параметрів мережі стає дуже повільним. Це називається зниклою проблемою градієнта.

У мережі шар ReLu розміщується після кожного згорткового та щільно з'єднаного шарів.

2.4.2 VGG16

Ця архітектура створена групою VGG з Оксфорда. Це покращує порівняно з AlexNet, замінюючи великі фільтри розміром ядра (11 і 5 у першому та другому

згорткових шарах відповідно) на кілька фільтрів розміром ядра 3X3 один за одним. З заданим сенсорним полем (ефективний розмір площі вхідного зображення, від якого залежить вихід), декілька складених ядер меншого розміру краще, ніж ядро з більшим розміром, оскільки кілька нелінійних шарів збільшують глибину мережі, що дозволяє їй вивчити більш складні функції з меншою кількістю параметрів.

Наприклад, три фільтри 3X3, задіяні послідовно, мають розмір сприйняття 7, але кількість задіяних параметрів становить 3 * (9C ^ 2) порівняно з 49C ^ 2 параметрами ядер розміром 7. Тут передбачається, що кількість вхідного і вихідного каналів шарів - це C. Крім того, ядра 3X3 допомагають зберігати активаційні карти більш високого рівня.

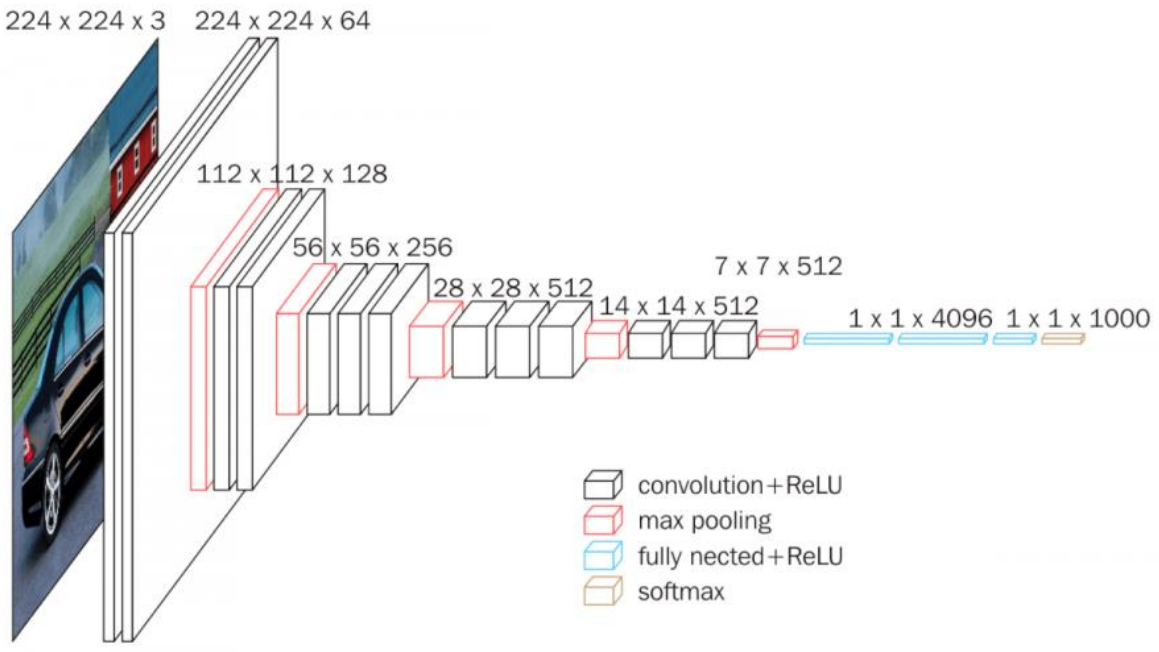

Рисунок 2.2 – Структура згорткової нейронної мережі AlexNet

2.4.3 GoogLeNet та Inception

Хоча VGG досягає феноменальної точності на наборі даних ImageNet, його розгортання навіть у самих скромних графічних процесорах є проблемою через величезні обчислювальні вимоги, як з точки зору пам'яті, так і часу. Він стає неефективним через велику ширину згорткових шарів.

Наприклад, для згортковий шару з розміром ядра 3X3, який приймає 512 каналів як вхід і виводить 512 каналів, кількість операцій становить 9 x 512 x 512 = 2359296.

При згортковій операції в одному місці кожен вихідний канал (512 у прикладі вище) підключається до кожного вхідного каналу, і тому ми називаємо це щільною архітектурою з'єднання. GoogLeNet ґрунтується на ідеї, що більшість активацій у глибокій мережі є або непотрібними, або зайвими через кореляції між ними. Тому найефективніша архітектура глибокої мережі матиме розріджений зв'язок між активаціями, що означає, що всі 512 вихідні канали не матимуть зв'язку з усіма 512 вхідними каналами. Існують методи вирізання таких з'єднань, які призводять до рідкої ваги / з'єднання. Але ядра для розрідженого множення матриць не оптимізовані в пакетах, що використовуються для навчання штучних нейронних мереж), що робить їх навіть повільніше, ніж їх щільні аналоги.

Таким чином, при розробці GoogLeNet було створено модуль, який називається, який наближає розріджену згорткову нейронну мережу до нормальної щільної конструкції. Оскільки лише невелика кількість нейронів ефективна, як згадувалося раніше, ширина / кількість згорткових фільтрів певного розміру ядра залишається невеликою. Крім того, він використовує звивини різного розміру для зйомки деталей у різних масштабах (5X5, 3X3, 1X1).

Ще одним важливим моментом щодо модуля є те, що він має так званий вузький. Це допомагає істотно скоротити необхідну кількість обчислень, як пояснено нижче.

Візьмемо перший прикладний модуль GoogLeNet, який має вхід 192 каналів. У ньому всього 128 фільтрів розміром ядра 3X3 та 32 фільтри розміром 5X5. Порядок обчислення для фільтрів 5X5 становить 25X32X192, який може продовжити збільшуватись, коли ми заглиблюємося в мережу, і ширина мережі та кількість фільтра 5X5 ще більше збільшуватимуться. Щоб уникнути цього, модуль початку використовує згортки 1X1 перед застосуванням ядер більшого розміру, щоб зменшити розмірність вхідних каналів, перш ніж подавати в ці згортки. Отже, у першому модулі створення вхід в модуль спочатку подається в 1X1 згортки всього 16 фільтрів, перш ніж він подається в 5X5 згортки. Це зменшує обчислення до 16X192 + 25X32X16. Всі ці зміни дозволяють мережі мати велику ширину і глибину мережі водночас.

Ще одна зміна, яку вніс GoogLeNet, полягала в тому, щоб замінити повністю з'єднані шари в кінці простим середнім глобальним об'єднанням, яке середнє значення каналів на двовимірній карті об'єктів після останнього згорткового шару. Це різко зменшує загальну кількість параметрів. Це можна зрозуміти з AlexNet, де щільно поєднанні шари містять близько 90% параметрів. Використання великої ширини та глибини мережі дозволяє GoogLeNet видаляти щільні шари, не впливаючи на точність. Завдяки цьому мережа досягає 93,3% точності на даних ImageNet і набагато швидше, ніж VGG.

2.4.4 Залишкові мережі

Відповідно до попередніх спостережень, збільшення глибини повинно підвищити точність мережі, доки не буде забезпечено перенасичення. Але проблема із збільшенням глибини полягає в тому, що сигнал, необхідний для зміни ваг, який виникає з кінця мережі, порівнюючи цільовий вихід і передбачення стає дуже малим на попередніх шарах, через збільшення глибини. Це по суті означає, що більш ранні шари навчаються майже незначно. Ця проблема, що виникає при тренуванні глибинних нейронних мереж, називається зниклим градієнтом. Інша проблема з навчанням глибших мереж полягає в проведенні оптимізації на величезному просторі параметрів і, таким чином, базове додавання шарів призводить до більш високої похибки тренування. Залишкові мережі дозволяють навчати такі глибокі моделі, будуючи мережу з модулей, які називаються залишковими шарами. Залишковий шар складається з частини, що відтворює класичну структуру двох послідовно з'єднаних згорткових шарів, і так званого «прямого зв'язку», що поєднує вхідний та вихідний шари, додаючи матрицю, отриману як вхідне значення до вихідної (рис. 2.3).

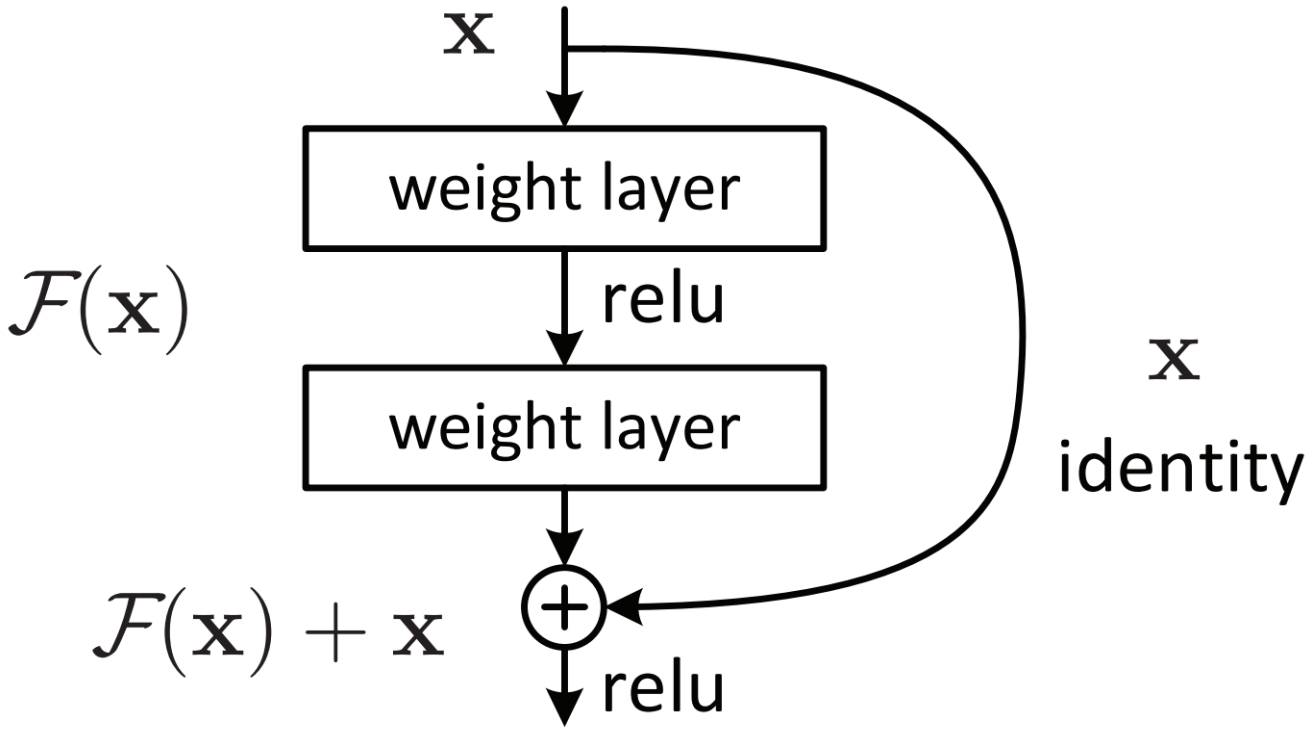

Рисунок 2.3 – Структура залишкового шару

Таким чином мережа має змогу на ранніх стадіях імітувати функцію ідентичності (у випадку, коли ваги гару наближені до 0), і відповідно модель отримує можливість набувати такого ж вигляду, як мережа меншої глибини. Після цього у процесі навчання реалізуються переваги глибшої мережі у вигляді набагато ширшого простору функцій, що можливо реалізувати.

2.5 Використання матриці Грама для аналізу споріднених текстур

Матрицею Грама системи векторів $e_1, e_2, \dots, e_n$ називається матриця, що визначена таким чином:

$$G(e_1, e_2, \dots, e_n) = \begin{pmatrix} (e_1, e_1) & \cdots & (e_1, e_n) \\ \vdots & \ddots & \vdots \\ (e_n, e_1) & \cdots & (e_n, e_n) \end{pmatrix} \quad (2.5.1)$$

В основі наведеного методу лежить обчислення квадратичної норми різниці матриць Грама цільового зображення та зображення стилю.

Інтуїтивно, можна вважати величину скалярного добутку векторів мірою їх схожості, масштабованою щодо норм векторів. Розглянемо два вектори, а саме випрямлені у вектор матриці кожного каналу, отриманого в активаційній карті нейронної мережі на певній глибині, що представляють певні особливості вхідного простору

Нехай шар нейронної мережі на глибині D має C каналів. Для кожного каналу представимо відповідну йому на активаційній карті двомірну матрицю у вигляді «випрямленого» вектора. Отримані вектори характеризують певну особливість вхідного простору, все ще зберігаючи просторову інформацію щодо того, у яких сегментах зображення ця особливість виявляється в більшій чи меншій мірі. Розглянемо усі отримані таким чином C векторів із активаційної карти та обчислимо скалярний добуток кожного з кожним (у тому числі із самим собою). Тепер ми можемо сформувати матрицю Грама (розміром CxC). . Це в певному сенсі дає інформацію про стиль (текстуру) зображення та жодної інформації щодо просторової структури, оскільки після виконання скалярного добутку отримана матриця жодним чином не пов’язано з позиційним розташуванням початкових значень активаційної карти.

Висновки до розділу 2

Штучні нейронні мережі надали можливість за допомогою алгоритму «тренування» розв’язувати оптимізаційні задачі, які були нерозв’язними для класичних методів. Серед таких моделей виділяється клас згорткових нейронних мереж, що показав високі результати у різноманітних задачах, пов’язаних з аналізом візуальної інформації. Протягом тренування таких мереж утворюються латентні репрезентації зображення у вигляді активаційних карт, що зберігають в собі семантичну інформацію різних рівнів. Аналіз зображення за допомогою активаційних карт надає можливість якісно порівнювати зображення на рівні як структур низького рівня, так і семантичного вмісту.

РОЗДІЛ 3 АРХІТЕКТУРА ТА АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ ПОБУДОВАНОГО ПРОГРАМНОГО ПРОДУКТУ

3.1 Вибір мови програмування

В якості мови реалізації програми було обрано мову програмування Python, а саме її версію Python 3. Основними характеристиками Python є:

- високорівневість – значний рівень абстракції коду;
- інтерпретовність – код аналізується построково, на відміну від мов, що вимагають компіляції всього коду перед запуском;
- загальне призначення – область застосування не обмежена специфічними задачами.

Python було створено голандським розробником Гідо ван Россумом та вперше представлено широкому доступу у 1991 році. Метою його створення була розробка нової мови, яка інтегрує функції більш ранньої мови ABC та нові функції, такі як гнучкість до користувацьких модифікацій та обробка виключень. Перша версія мови, Python 1.0 була випущена у 1994 році. Вона запозичила модульну систему від Modula-3, мала можливість взаємодіяти з операційною системою Amoeba і була побудована відповідно до принципів функціонального програмування.

Принципи мови Python полягають у акценті на читабельність та зрозумілість коду. Його мовні конструкції та об'єктно-орієнтований підхід мають на меті створення чіткий логічний код як для малих, так і для масштабних проектів. Python розроблений за ліцензією відкритого коду, затвердженою OSI, що робить його

вільним для використання та розповсюдження, у тому числі для комерційного використання.

Ключовими особливостями Python є динамічна типізація (типи змінних не є завчасно зафіксованими, а визначаються у процесі інтерпретації) і так званий «збирач сміття». Python підтримує декілька парадигм програмування, включаючи структуроване (зокрема, процедурне), об'єктно-орієнтоване та функціональне програмування.

Деякі з переваг програмування мовою Python:

1. Наявність сторонніх модулів:

Індекс пакунків Python (PyPI) містить численні сторонні модулі, які роблять Python здатним взаємодіяти з більшістю інших мов та платформ.

2. Бібліотеки розширеної підтримки:

Python пропонує велику стандартну бібліотеку, що охоплтює, як Інтернет-протоколи, струнні операції, інструменти веб-служб та інтерфейси операційної системи. Багато завдань програмування з високим рівнем використання вже написані у стандартній бібліотеці, що значно скорочує довжину коду, який потрібно записати.

3. Відкритий код та розробка спільнотою:

Мова Python розроблена під затвердженою OSI ліцензією з відкритим кодом, що дозволяє вільно використовувати та розповсюджувати, в тому числі для комерційних цілей.

Крім того, його розвиток визначається спільнотою, яка учасники якої регулярно додають, перевіряють код та публікують пояснення та документацію.

4. Легкість навчання та доступна підтримка:

Python пропонує відмінну читабельність і зрозумілий синтаксис, простий у навчанні, що допомагає новачкам використовувати цю мову програмування.

Настанови щодо стилю коду, PEP 8, містять набір правил для полегшення форматування коду. Крім того, наявність великої спільноти користувачів та активних розробників призвела до того, що багатий Інтернет-ресурсний банк сприяв розвитку та подальшому прийняттю мови.

5. Структури даних:

Python має вбудовані структури даних, такі як список та словник, які можна використовувати для побудови швидких для виконання структур даних. Крім того, Python також надає можливість динамічного додання даних на високому рівні, що зменшує тривалість виконання коду.

6. Продуктивність та швидкість:

Python має чистий об'єктно-орієнтований дизайн, забезпечує розширені можливості управління процесами, володіє сильними можливостями інтеграції та обробки тексту та власними засобами тестування одиниць, що сприяє підвищенню швидкості та продуктивності. Python вважається життєздатним варіантом для створення складних багатопротокольних мережевих додатків.

3.2 Використання бібліотек та модулів

В рамках реалізації програмного продукту було використано ряд бібліотек та модулів для мови python. Нижче представлено опис тих з них, що є ключовими для роботи програми.

3.2.1 Вбудовані модулі

os – модуль, що надає розробнику можливість використовувати функції взаємодії з операційною системою. належить до стандартних модулів-утиліт Python. Цей модуль пропонує портативний спосіб використання функціоналу, що є специфічним для операційної системи.

sys – модуль, що забезпечує доступ до деяких змінних, використовуваних або підтримуваних інтерпретатором, та до функцій, які сильно взаємодіють з інтерпретатором.

io – модуль, що забезпечує основні засоби Python для роботи з різними типами вводу-виводу. Існує три основні типи вводу / виводу: текстове введення / виведення, двійкове введення / виведення та введення / виведення сирих даних. Це універсальні категорії, і для кожної з них можуть використовуватися різні механізми зберігання. Конкретний об'єкт, що належить до будь-якої з цих категорій, називається файловим об’єктом. Інші поширені терміни - це об'єкт, що нагадує потік та файл. Незалежно від своєї категорії, кожен об'єкт конкретного потоку також матиме різні можливості: він може бути лише для читання, лише для читання або для читання-запису. Він також дозволяє довільний випадковий доступ (пошук вперед або назад до будь-якого місця) або лише послідовний доступ (наприклад, у випадку сокету (програмного інтерфейсу для забезпечення обміну даними між процесами) або конвеєру програмного забезпечення).

3.2.2 Python Imaging Library

Python Imaging Library (скорочено PIL) (у нових версіях, відомий як Pillow) - це безкоштовна та відкрита додаткова бібліотека для мови програмування Python, яка додає підтримку для відкриття, редагування та збереження багатьох різних форматів файлів зображень. Він доступний для Windows, Mac OS X та Linux. Остання версія PIL 1.1.7, випущена у вересні 2009 року та підтримує Python 1.5.2-2.7. Оскільки розробку основного проекту було припинено, було створено новий модуль під назвою Pillow, що оновлюється спільнотою розробників з метою додавання нового функціоналу та забезпечення сумісності з Python 3.

Pillow пропонує кілька стандартних процедур для маніпулювання зображенням. До них належать:

- внесення піксельних змін,
- маскування та обробка прозорості,
- фільтрація зображень, у тому числі розмивання, згладжування контурів та пошук меж,
- поліпшення якості зображення, наприклад збільшення різкості, регулювання яскравості, контрасту чи кольору,
- додавання тексту до зображень та багато іншого.

3.2.3 PyQt5

PyQt5 – це модифікація бібліотеки графічного інтерфейсу користувача Qt, що надає доступ до її функціоналу за допомогою команд мовою Python. Qt - це бібліотека із відкритим кодом, що складається з інструментів створення та керування для елементів-складових, на основі яких утворюються графічні інтерфейси користувача, а також крос-платформних додатків. Інтерфейси працюють на різних програмних та апаратних платформах, таких як Linux, Windows, macOS, Android або вбудованих системах з незначною або без змін базової кодової бази, при цьому все ще є нативним додатком із власними можливостями та високою швидкістю.

Qt побудований на ключових поняттях, що наведені нижче.

Повна абстракція графічного інтерфейсу - перші версії Qt використовували власний механізм відображення графіки та користувацької взаємодії, імітуючи вигляд різних платформ, на яких він працює, коли створює графічні елементи. Це полегшило роботу з крос-платформним перенесенням, оскільки дуже мало класів у Qt справді залежало від цільової платформи; однак це час від часу призводило до невеликих розбіжностей, коли ця емуляція була недосконалою. Останні версії Qt використовують вбудовані стилі API різних платформ, на платформах, які мають нативні графічні елементи, для запиту метрик та відображення більшості елементів керування, і такі розбіжності фактично не виникають. На деяких платформах Qt - це власний API. Деякі інші портативні графічні набори інструментів прийняли різні архітектурні рішення; наприклад, wxWidgets використовує набори інструментів цільової платформи для своїх реалізацій.

Сигнали та слоти - мовна конструкція, запроваджена в Qt для зв'язку між об'єктами, що полегшує реалізацію шаблону спостерігача, уникаючи кодового коду. Концепція полягає в тому, що елементи GUI можуть передавати сигнали, що

містять інформацію про події, які можуть бути отримані іншими елементами управління, використовуючи спеціальні функції, відомі як слоти.

Компілятор метаоб'єктів, що називається moc - це інструмент, який запускається на джерелах програми Qt. Він інтерпретує певні макроси з коду C ++ як анотації та використовує їх для створення доданого коду C ++ з метаінформацією про класи, що використовуються в програмі. Ця мета-інформація використовується Qt для надання функцій програмування, недоступних в C ++: сигналів і слотів, самоаналізу та асинхронних викликів функцій.

Qt може використовуватися в кількох інших мовах програмування, таких як Python, Javascript, C # або Rust за допомогою мовного зв’язування даних.

3.2.4 Numpy

Numpy – це бібліотека для Python, яка додає підтримку великих, багатовимірних масивів і матриць. з великою колекцією математичних функцій високого рівня для роботи над цими масивами. Родоначальник NumPy, Numeric, спочатку був створений Джимом Гугуніним за участі кількох інших розробників. У 2005 році Travis Oliphant створив NumPy, включивши в Numarray функції з Numeric зі значними модифікаціями. NumPy є програмним забезпеченням з відкритим кодом та має велику спільноту розробників, що працюють над його просуванням.

NumPy націлений на реалізацію еталонної програми Python, яка є оптимізатором інтерфейсу байт-коду. Математичні алгоритми, написані для цієї версії Python, часто працюють набагато повільніше, ніж складені еквіваленти. NumPy частково вирішує проблему повільності, надаючи багатовимірні масиви та

функції та операторів, які ефективно працюють над масивами, вимагаючи переписати деякий код, переважно внутрішні цикли за допомогою NumPy.

Використання NumPy в Python дає функціональність, порівнянну з MATLAB, оскільки вони обидва інтерпретуються, і вони обидва дозволяють користувачеві писати швидкі програми, якщо більшість операцій працює над масивами чи матрицями замість скалярів. Для порівняння, MATLAB може похвалитися великою кількістю додаткових наборів інструментів, зокрема Simulink, тоді як NumPy суттєво інтегрований з Python, більш сучасною і повною мовою програмування. Крім того, доступні додаткові пакети Python; SciPy - це бібліотека, яка додає більше функцій, схожих на MATLAB. Внутрішні функції MATLAB і NumPy покладаються на пакети BLAS і LAPACK для ефективних обчислень лінійної алгебри.

Функції широко використовуваної бібліотеки комп'ютерного зору OpenCV переважно використовують масиви NumPy для зберігання та роботи з даними. Оскільки зображення з декількома каналами просто представляти у вигляді тривимірних масивів, індексація, нарізка або маскування за допомогою інших масивів – це дуже ефективні способи доступу до певних пікселів зображення. Масив NumPy як універсальна структура даних у OpenCV для зображень, вилучених точок функцій, ядер фільтру та багатьох інших призначень значно спрощує робочий процес програмування та подільшого налагодження.

3.2.5 Pytorch

PyTorch - це бібліотека машинного навчання з відкритим кодом, заснована на бібліотеці Torch,яка використовується для таких додатків, як комп'ютерне бачення

та обробка природніх мов, розроблена насамперед лабораторією AI Research (FAIR) Facebook. Це безкоштовне та відкрите програмне забезпечення, що випускається під ліцензією Modified BSD. Хоча інтерфейс Python є більш відполірованим та основним напрямком розвитку, PyTorch також має інтерфейс для мови C ++.

На PyTorch побудовано ряд програм програм Deep Learning, включаючи Pyro Uber, Трансформери HuggingFace, та Catalyst.

PyTorch надає дві функції високого рівня: тензорні обчислення (наприклад, NumPy) з сильним прискоренням за допомогою графічних процесорів (GPU), та можливість застосування глибинних нейронних мереж, побудованих на основі стрічкової системи автоматичної диференціації

$$(h \star w)(x) = \sum_{k=0}^{k=N} h(x+k)w(k) \qquad (3.2.5)$$

3.3 Архітектура програми

В основі програми лежить застосування мережі VGG16 для отримання активаційних карт із шарів, глибина яких задається користувачем. Користувач задає два набори значень глибин – для семантичних шарів, за допомогою яких обчислюється складова функції втрат з вхідним зображенням, і ті, для яких обчислюється матриця Грама та порівнюється з відповідною матрицею зображення стилю. Також задається кількість кроків оптимізації функції втрат, що виконуються за допомогою методу зворотного розповсюдження градієнтів, та коефіцієнт α, який домножається на складову функції втрат, що відповідає за стиль.

Сама функція втрат набуває такого вигляду:

$$L = \sum_{l \in M_{\text{семант}}} \beta_l L^l_{\text{семант}} + \alpha \sum_{l \in M_{\text{стилю}}} \gamma_l L^l_{\text{стилю}} \quad (3.3.1)$$

Тут $L^l_{\text{семант}}$ – семантична функція втрат для шару l, $L^l_{\text{стилю}}$ – функція втрат стилю для шару l, $M_{\text{семант}}$ і $M_{\text{стилю}}$ – відповідні множини шарів, задані користувачем, α – зважуючий коефіцієнт, також заданий користувачем, β_l і γ_l – зважуючі коефіцієнти, що в межах програми приймають значення 1, але можуть бути налагоджені для налаштування більш тонкого балансу між змістом і стилем отриманого зображення.

При цьому семантична функція втрат визначена таким чином:

$$L^l_{\text{семант}} = \sum_{i,j} \left(A^l_{ij}(g) - A^l_{ij}(c) \right)^2, \quad (3.3.2)$$

де g – матриця цільового зображення,

c – матриця вхідного зображення,

A^l – матричний оператор, що отримує активаційну карту зображення глибини l (композиція функцій, заданих шарами нейронної мережі).

Функція втрат стилю визначається так:

$$L^l_{\text{стилю}} = \frac{1}{S_l} \sum_{i,j} \left(G^l_{ij}(g) - G^l_{ij}(s) \right)^2, \quad (3.3.3)$$

де g – матриця цільового зображення,

c – матриця зображення стилю,

S_l – є добутком двох перших розмірностей активаційної карти,

G^l – матриця Грама, побудована на активаційній карті глибини l.

$$G_{ij}^l(x) = \sum_k A_{ik}^l(x) A_{jk}^l(x) \quad (3.3.4)$$

3.4 Інтерфейс програми

Інтерфейс програми написано за допомогою бібліотеки PyQt. Користувачу надається можливість вибрати два файли на своєму пристрої, що будуть відігравати відповідно роль вхідного зображення та зображення стилю. Права панель надає можливість котролювати параметри – індекси шарів, що використовуються у функції втрат, розмір кроку оптимізації та їх загальна кількість, роздільна здатність отриманого зображення, зважуючий коефіцієнт функції втрат та частота, з якою цільове зображення буде оновлюватись в процесі оптимізації. Також додано трекер прогресу, що дозволяє побачити, на якій стадії оптимізації знаходиться програма у процесі виконання оптимізації. Інтерфейс програми у повноекранному режимі відображено на рисунку 3.1.

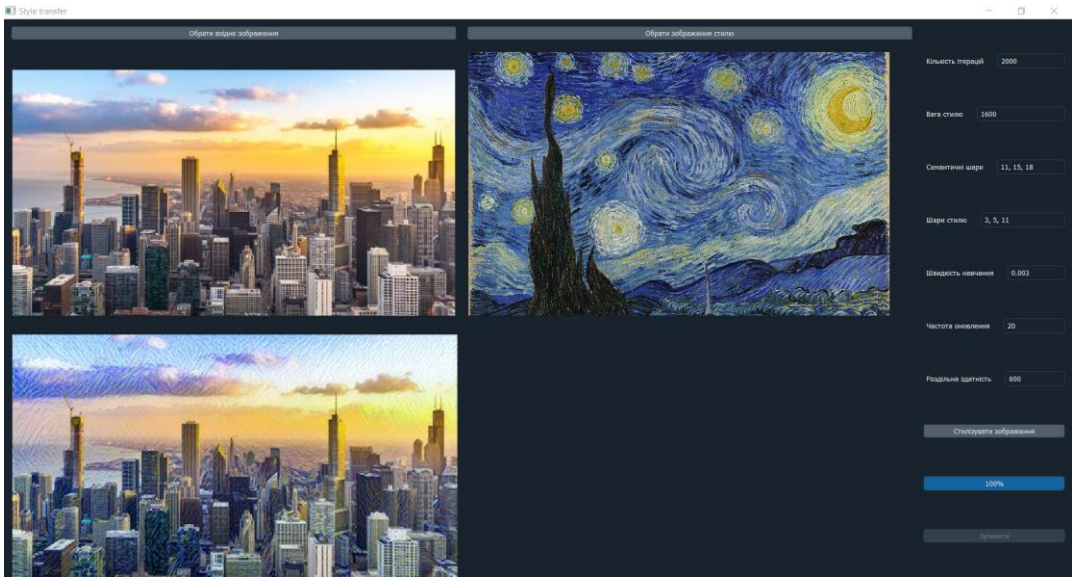

Рисунок 3.1 – Інтерфейс програми

3.5 Виконані експерименти

Для підтвердження ефективності описаного методу в рамках роботи було проведено декілька експериментів, в яких перенесення стилю застосовується до різних початкових та цільових зображень

3.5.1 «Зоряна ніч»

Для першого експерименту було взято за основу зображення міста під час заходу сонця, в якості зображення стилю – картину Ван Гога «Зоряна ніч» (що є класичним прикладом для виконання перенесення стилю, оскільки має яскраво

виражену текстуру). Вхідні зображення наведені на рисунках 3.2 і 3.3, а результати виконання можна побачити на рисунку 3.4, отриманому знімком екрану після виконання програми.

Рисунок 3.2 – Вхідне зображення

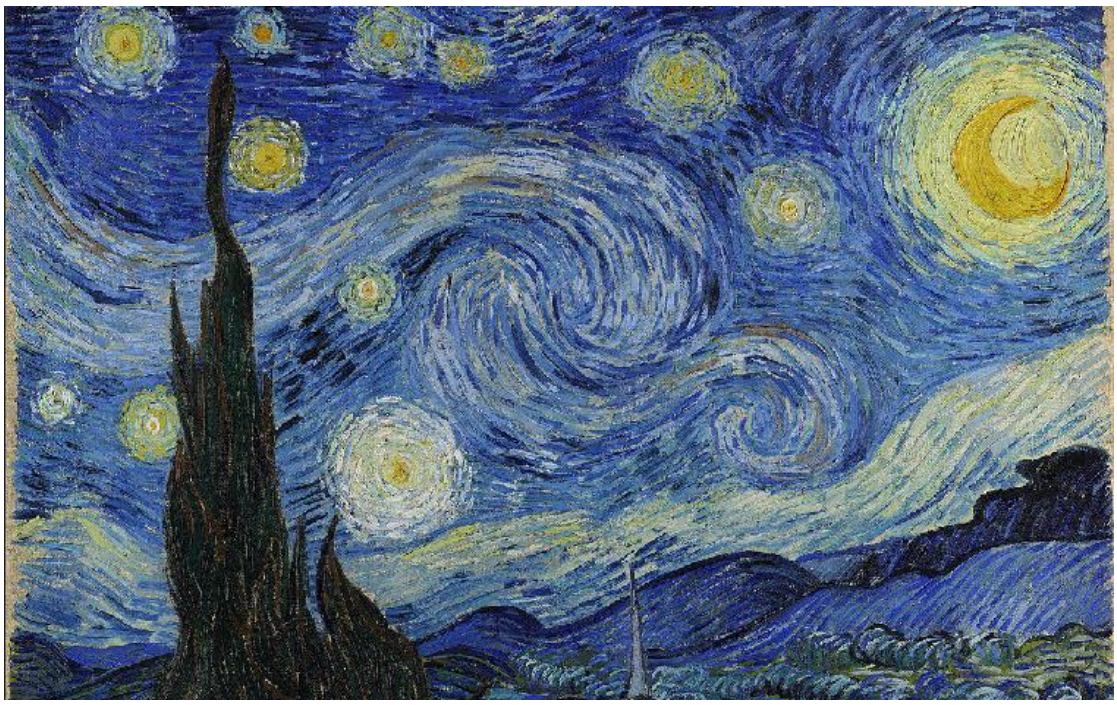

Рисунок 3.3 – Зображення стилю

Рисунок 3.4 – Отримане зображення

3.5.2 Текстура тканини

Для другого експерименту було взято за основу фотографію-портрет, в якості зображення стилю – зображення тканини. Вхідні зображення наведені на рисунках 3.5 і 3.6, а результати виконання можна побачити на рисунку 3.7.

Рисунок 3.5 – Вхідне зображення

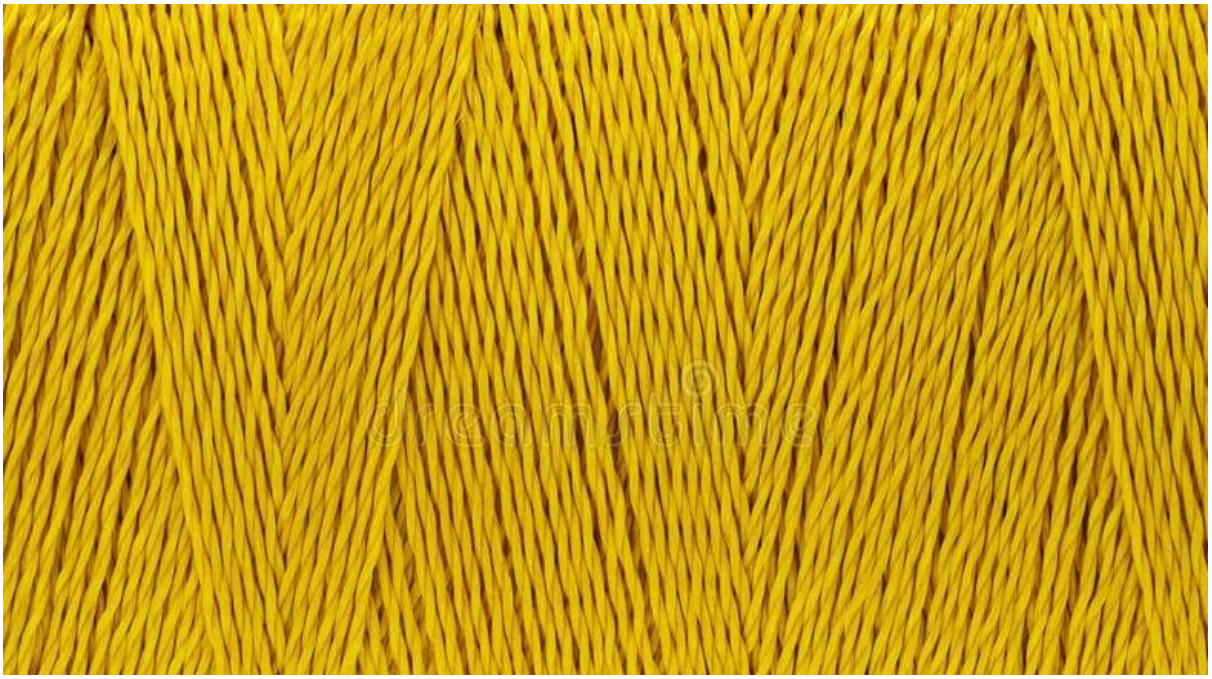

Рисунок 3.6 – Зображення стилю

Рисунок 3.7 – Отримане зображення

Висновки до розділу 3

В межах роботи було розроблено і продемонстровано систему перенесення стилю, з наявністю значної кількості параметрів алгоритму, що задаються користувачем за допомогою графічного інтерфейсу.

РОЗДІЛ 4: ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

4.1 Постановка задачі проектування

Спроектувати програмний продукт для перенесення стилю зображень. Експертна система розроблена за допомогою мови програмування python.

4.2 Обґрунтування функцій та параметрів програмного продукту

Головна функція F_0 — розробка програмного продукту, функція якого полягає у перенесенні стилю одного зображення на інше. Виходячи з конкретної мети, можна виділити наступні основні функції ПП:

1) F_1 — вибір мови програмування;
2) F_2 — вибір фреймворку для реалізації графічного інтерфейсу;
3) F_3 — вибір базової архітектури нейронної мережі.

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція F_1:

1) мова програмування python;
2) мова програмування C++.

Функція F_2:

1) використання фреймворку tkinter;

2) використання фреймворку Qt.

Функція F_3:

1) використання архітектури VGG;

2) використання архітектури MobileNet.

Виходячи з представлених варіантів будуємо морфологічну карту (рис.4.1).

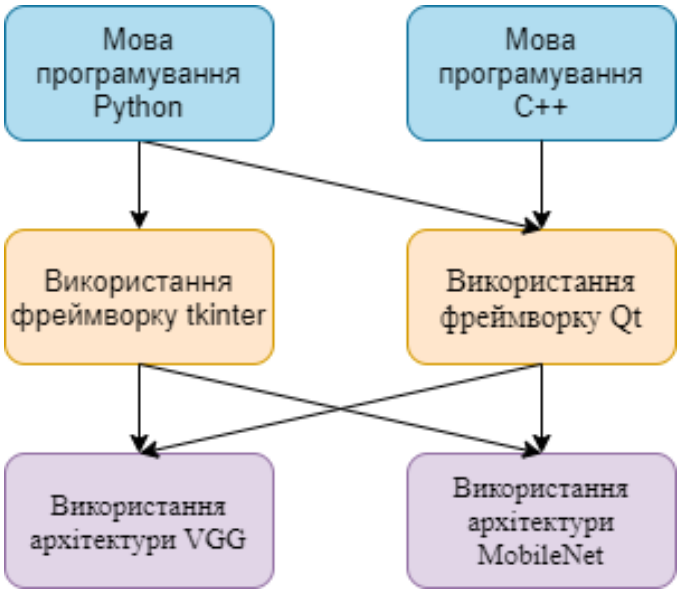

Рис. 4.1 Морфологічна карта

Спираючись на карту була побудована позитивно-негативна матриця(табл. 4.1)

Таблиця 4.1– Позитивно-негативна матриця

Основні функції	Варіанти реалізації	Переваги	Недоліки

F1	А	Наявність високорівневих бібліотек, можливість вільно модифікувати програму	Менша швидкість виконання програми
	Б	Можливість оптимізувати код, пришвидшуючи виконання програми	Необхідність повністю відтворювати модулі мережі
F2	А	Менш об'ємна програма	Обмежена кількість інструментів реалізації графічних елементів і взаємодії з функціоналом
	Б	Доступ до більшої кількості графічних елементів та гнучкіша взаємодія з рештою коду	Наявність великої кількості додаткового коду, складніше знаходити збої
F3	А	Якісне представлення зображення на активаційних картах	Велика кількість параметрів і об’єм оперативної пам’яті
	Б	Менший час виконання, малий об’єм оперативної пам’яті	Менш інформативний латентний простір активаційних карт

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція *F1:*

Оскільки створення цілісного програмного продукту мовою C++ потребувало б написання набагато більш об’ємного і при цьому вузькоспеціалізованого коду, що обмежило б можливість проводити експерименти і розвивати продукт. Тому використовуємо варіант А.

Функція *F2:*

Бібліотека графічного користувацького інтерфейсу tkinter, не зважаючи на певні переваги, не має достатнього функціоналу для повноцінного забезпечення роботи програми, тому зупиняємо вибір на варіанті Б.

Функція *F3:*

Обидва варіанти мають свої переваги та недоліки, необхідним є більш детальне порівняння.

Таким чином, будемо розглядати такі варіанти реалізації ПП:

- F1А – F2Б – F3А
- F1А – F2Б – F3Б

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

– *X1* – Час розробки;

– *X2* – Час виконання алгоритму;

– *X3* – Затримка між взаємодією з інтерфейсом і виконанням алгоритму;

– *X4* – Необхідний ресурс пам'яті відеокарти.

– *X5* – Необхідний ресурс оперативної пам’яті.

Можемо встановити такі співвідношення між функціями ПП та наведеними параметрами:

F1 відповідають X1 і X2, F2 – X3, F3 – X4 і X5.

Для характеристики прототипу програмного додатку використовуємо

параметри X1 – X5. На основі даних, що представлені у літературі, визначаємо мінімальні, середні отримуванні та максимально допустимі значення (табл. 4.2)

Таблиця 4.2 – Система параметрів додатку

Найменування параметру	Позначення параметру	Значення параметру		
		Мінімальне	Середнє	Максимальне
Час розробки, людина*год	X1	60	90	120
Час виконання алгоритму, мс	X2	800	1300	1800
Затримка між взаємодією з інтерфейсом і виконанням алгоритму, мс	X3	80	110	140
Необхідний ресурс пам'яті відеокарти, МБ	X4	1600	2000	2400
Необхідний ресурс оперативної пам’яті, МБ	X5	1100	1250	1400

За даними таблиці 4.2 будуються графічні характеристики параметрів, наведені на рисунках 4.2 – 4.6.

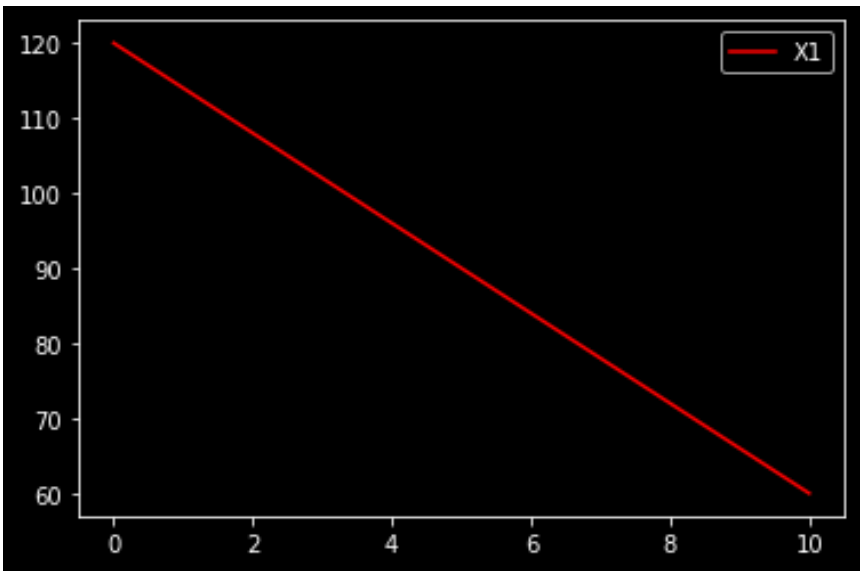

Рисунок 4.2 – Значення параметру X1, Час розробки

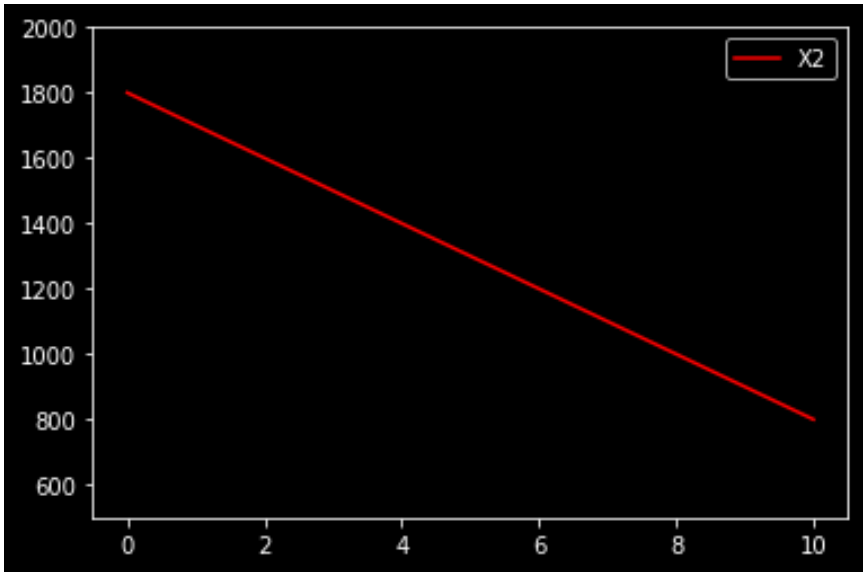

Рисунок 4.3 – Значення параметру X2, Час виконання алгоритму

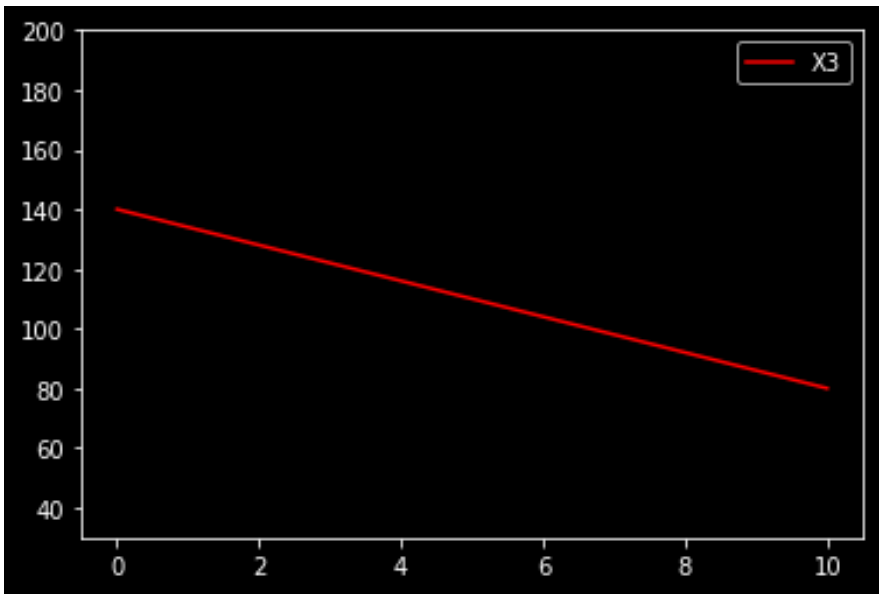

Рисунок 4.4 – Значення параметру X3, Затримка між взаємодією з інтерфейсом і виконанням алгоритму

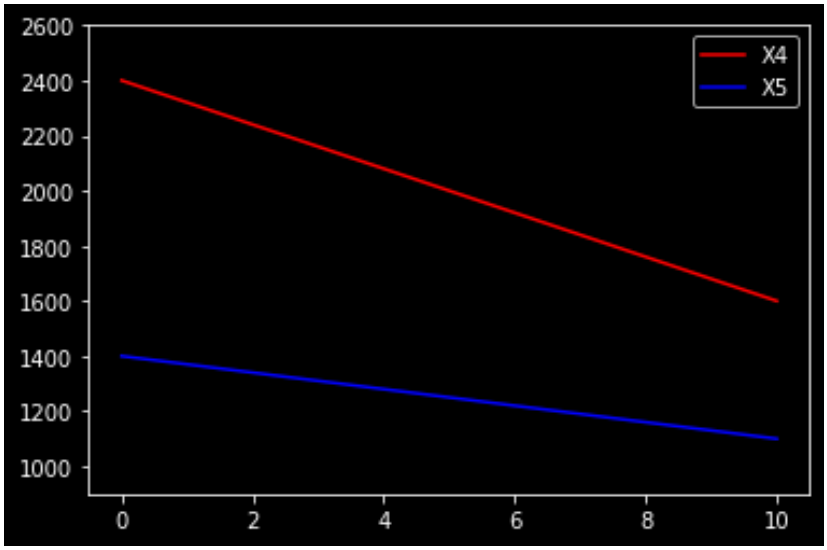

Рисунок 4.5 – Значення параметрів X4, Необхідний ресурс пам'яті відеокарти, і X5, Необхідний ресурс оперативної пам’яті

Вагомість параметрів оцінюється за допомогою методів попарного зрівняння. Кожному параметру відповідно до оцінки експерта встановлюється у відповідність ранг від 1 до 5, де 5 – найменш вагомий, 1 – найбільш вагомий. Результати наведені в таблицях 4.3-4.4.

Таблиця 4.3 Результат оцінки параметрів

Параметр	Ранг параметру по оцінці експерта							Сума рангів, R_i	Відхилення Δ_i	Квадрат відхилення, $(\Delta_i)^2$
	1	2	3	4	5	6	7			
X1	2	1	1	1	2	1	3	11	10	100
X2	1	2	2	3	1	2	1	12	9	81
X3	5	3	4	5	3	3	4	27	-6	36
X4	4	5	5	4	5	5	5	33	-12	144
X5	3	4	3	2	4	4	2	22	-1	1
Разом	15	15	15	15	15	15	15	105	0	362

Таблиця 4.4 – Попарне порівняння параметрів

Параметри	**Експерти**							**Кінцева оцінка**	**Числове значення**
	1	**2**	**3**	**4**	**5**	**6**	**7**		
X1 та X2	<	>	>	>	<	>	<	>	1.5
X1 та X3	>	>	>	>	>	>	>	>	1.5
X1 та X4	>	>	>	>	>	>	>	>	1.5
X1 та X5	>	>	>	>	>	>	<	>	1.5
X2 та X3	>	>	>	>	>	>	>	>	1.5
X2 та X4	>	>	>	>	>	>	>	>	1.5
X2 та X5	>	>	>	<	>	>	>	>	1.5
X3 та X4	<	>	>	<	>	>	>	>	1.5
X3 та X5	<	>	<	<	>	>	<	<	0.5
X4 та X5	<	<	<	<	<	<	<	<	0.5

Порахуємо коефіцієнт конкордації:

$$W = \frac{12S}{N^2(n^3-n)} = \frac{12 \cdot 362}{7^2(5^3-5)} = 0{,}739 > W_k = 0{,}67 \qquad (4.2.1)$$

Так як коефіцієнт конкордації більше нормативного, результати вважають достовірними.

Розрахунок вагомості параметрів наведено в табл. 4.5

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметри	Параметри Xj					Перший крок		Другий крок		Третій крок	
	X1	X2	X3	X4	X5	b_i	$К_{вi}$	b_i	$К_{вi}$	b_i	$К_{вi}$
X1	1	1,5	1,5	1,5	1,5	7	0.28	49	0.363	343	0.443
X2	0,5	1	1,5	1,5	1,5	6	0.24	36	0.267	216	0.279
X3	0,5	0,5	1	1,5	0,5	4	0.16	16	0.119	64	0.083
X4	0,5	0,5	0,5	1	0,5	3	0.12	9	0.067	27	0.035
X5	0,5	0,5	1,5	1,5	1	5	0.2	25	0.185	125	0.161
Загалом:						25	1	135	1	775	1

Враховуючи дані з порівнянь варіантів реалізацій функцій можна виключити з реалізацій функцій наступні варіанти: F1(Б), F2(А).

Залишаються наступні варіанти:

1. F1(А) => F2(Б) => F3(А)
2. F1(А) => F2(Б) => F3(Б)

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основна функція	Варіант реалізації	Параметр	Абсолютне значення параметру	Бальна оцінка параметру	Коефіцієнт вагомості параметру	Коефіцієнт якості
F1	А	X1	105	2.5	0.443	1.108
		X2	1585	2.15	0.279	0.6
F2	Б	X3	90	8.33	0.083	0.691
F3	А	X4	2230	2.13	0.035	0.075

		X5	1360	1.33	0.161	0.214
	Б	X4	1950	5.63	0.035	0.197
		X5	1220	6.0	0.161	0.966

Обрахуємо коефіцієнти якості кожного з варіантів розробки:

$$K_{я1} = 1.108 + 0.6 + 0.691 + 0.966 + 0.214 = 3.579$$

$$K_{я2} = 1.108 + 0.6 + 0.691 + 0.197 + 0.075 = 2.671$$

Оскільки варіант 1 має найбільший коефіцієнт якості, він є найкращим.

4.3 Економічний аналіз варіантів розробки

Для оцінки трудомісткості розробки спочатку проведемо розрахунок трудомісткості. Усі варіанти мають наступні основні завдання:

Всі варіанти включають в себе два окремих завдання: створення прототипу програмного продукту і написання коду програмного продукту.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

При цьому варіант 1 має додаткове завдання: обробка даних збереженої мережі VGG.

Варіант 2 має інше додаткове завдання: створення коду для відтворення архітектури за оригінальною академічною статтею.

Завдання 1 за ступенем новизни відноситься до групи В, завдання 2 — до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 — до групи 3.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 3, трудомісткість дорівнює: $T_Р$ = 23 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П$ = 1,2. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх чотирьох завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці другого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.7. Тоді загальна трудомісткість виконання першого завдання дорівнює:

$$T_1 = 23 \cdot 1.2 \cdot 1 \cdot 0.7 = 19.32 \text{ людино-дня.}$$

Для другого завдання (використовується алгоритм першої групи складності, степінь новизни Б), тобто $T_Р$ =50 людино-днів, $К_П$ = 0.8, $К_{СК}$ = 1, $К_{СТ}$ = 0.9:

$$T_2 = 50 \cdot 0{,}8 \cdot 1 \cdot 0{,}9 = 36 \text{ людино-днів.}$$

Для завдання 3.1 (використовується алгоритм другої групи складності, ступінь новизни – В):

Тр = 17 людино-днів; Кп = 0.6; Кст = 0.7; $T_{3.1}$ = 17 • 0.6 • 0,7 = 7.14.

Для завдання 3.2 (використовується алгоритм першої групи складності, ступінь новизни В):

Тр = 26 людино-днів; Кп = 0.8; Кст = 0,6; T_4 = 26 • 0.8 • 0,6 = 12,48.

Визначимо повну трудомісткість варіантів(людино-днів):

T_I = 19.32 + 36 + 7.14 = 62.46 людино-дня;

T_{II} = 19.32 + 36 + 12.48 = 67,8 людино-дня;

Найбільш високу трудомісткість має варіант II.

Далі вважається, що робочий день складає 8 годин, в тижні п’ять робочих днів. В розробці бере участь програміст з окладом 25000 грн та тестувальник з окладом 12000 грн. Визначимо середню заробітну плату за годину:

$$C_ч = \frac{25000+12000}{2*22*8} = 105.11 \quad (4.3.1)$$

Тоді заробітна плата для кожного з варіантів реалізації(грн):

1) $C_{ЗП} = 105.11 * 8 * 62.46 = 52521,37$
2) $C_{ЗП} = 105.11 * 8 * 67,8 = 57011,66$

Відрахування на соціальне страхування(22%)(грн):

1) $С_{ВІД} = 52521,365*0,22 = 11554,7$
2) $С_{ВІД} = 57011,664*0,22 = 12542,57$

Далі розрахуємо витрати на оплату однієї машино-години. Враховуючи, що вона обслуговує одного спеціаліста з окладом 25000 грн та одного з окладом 12000 грн з коефіцієнтом зайнятості 0,6, то для двох машин отримаємо

$$С_г = 12 * 25000 * 0,4 + 12 * 12000 * 0,4 = 266400 \text{ грн}$$

Враховуючи додаткову заробітну плату

$$С_{ЗП} = 266400 * (1 + 0,35) = 359640$$

Відрахування на соціальне страхування

$$С_{ВІД} = 359640 * 0,22 = 79120.8$$

Розрахуємо амортизаційні підрахунки

$$С_А = К_{ТМ} * К_А * Ц_{ПР} = 1,15 * 0,25 * 26000 = 7475 \text{ грн}$$

Розрахуємо витрати на ремонт та профілактику:

$$С_Р = К_{ТМ} * Ц_{ПР} * К_Р = 1,15 * 26000 * 0,05 = 1495 \text{ грн} \quad (4.3.2)$$

Розрахуємо ефективний годинний фонд часу ПК за рік

$$Т_{ЕФ} = (365 - 142 - 16) * 8 * 0,8 = 1324,8 \text{ год}$$

Розрахуємо витрати на електроенергію

$$С_{ЕЛ} = 1324,8 * 0,6 * 1,75 * 3 = 4173.12 \text{ грн}$$

Накладні витрати рівні:

$$С_Н = 26000 * 0,67 = 17420 \text{ грн.}$$

Отже експлуатаційні витрати(грн):

$$С_{ЕКС} = 359640 + 79120.8 + 7475 + 1495 + 4173.12 + 17420 = 469323.92$$

Тоді собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М-Г} = \frac{469323.92}{1324,8} = 354.26 \text{ грн/год} \quad (4.3.3)$$

Враховуючи, що всі роботи ведуться на ЕОМ, витрати на оплату машинного часу:

1) $С_м = 354.26 * 8 * 62,46 = 177016.63$
2) $С_м = 354.26 * 8 * 67.8 = 192150.62$

Накладні витрати відповідно

1) $С_Н = 177016.63 * 0,67 = 118601.14$
2) $С_Н = 192150.62 * 0,67 = 128740.92$

Розрахуємо повну вартість розробки за варіантами:

1) $С_{ПП} = 52521,37 + 11554,7 + 177016.63 + 118601.14 = 359693.84$
2) $С_{ПП} = 57011,66 + 12542,57 + 192150.62 + 128740.92 = 390445.8$

4.4 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня

$$К_{ТЕР1} = \frac{3,579}{359693.84} = 9,95 * 10^{-6} \quad (4.4.1)$$

$$К_{ТЕР2} = \frac{2,671}{390445.77} = 6,84 * 10^{-6} \quad (4.4.2)$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1}= 9{,}95 * 10^{-6}$.

Висновки до розділу 4

Отже враховуючи всі дослідження, що описані вище, можна сказати, що 1 варіант реалізації є найбільш оптимальним зі сторони якісно-економічної оцінки. Його коефіцієнт техніко-економічного рівня складає $9{,}95 * 10^{-6}$.

Розробка цього варіанту передбачає такі обов’язкові завдання як:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки.

3.1. Обробка даних збереженої мережі VGG

Даний варіант виконання експертної системи надає можливість досягти оптимального балансу швидкодії, зручності програмного продукту і необхідних ресурсів на його створення.

ВИСНОВКИ

Перенесення стилю – задача, яку, незважаючи на інтуїтивно просте розуміння людиною, складно задати алгоритмічно. Останні часом найбільш перспективні методи застосовують згорткові глибинні нейронні мережі для розв’язання цієї проблеми.

В межах роботи було розроблено і продемонстровано систему перенесення стилю, з наявністю значної кількості параметрів алгоритму, що задаються користувачем за допомогою графічного інтерфейсу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A. Krizhevsky, I. Sutskever, G. E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks // University of Toronto, 2012.
2. K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.
3. C. Szegedy et al., "Going deeper with convolutions," 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, 2015, pp. 1-9, doi: 10.1109/CVPR.2015.7298594.
4. Y. Jing, Y. Yang, Z. Feng, J. Ye, Y. Yu and M. Song, "Neural Style Transfer: A Review," in IEEE Transactions on Visualization and Computer Graphics, doi: 10.1109/TVCG.2019.2921336.
5. Gatys, Leon A., et al. "Controlling perceptual factors in neural style transfer." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017.
6. Li, Yijun, et al. "Universal style transfer via feature transforms." Advances in neural information processing systems. 2017.
7. Chen, Dongdong, et al. "Stylebank: An explicit representation for neural image style transfer." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.
8. Yanyan Zhao, Bing Qin and Ting Liu Integrating intra- and inter-document evidences for improving sentence sentiment classification. ACTA AUTOMATICA SINICA, 36. 2010. P.1417– 1425.
9. V. Hatzivassiloglou, K. McKeown Predicting the semantic orientation of adjectives. Proceedings of the Joint ACL/EACL Conference. 1997. P. 174–181.

10. T. Young, D. Hazarika, S. Poria, E. Cambria Recent trends in deep learning based natural language processing. IEEE Comput. Intell. Mag. Aug. 2018. Vol. 13, No. 3. P. 55–75.

11. Mikolov T., Karafiat M., Burget L., Cernock´y J., Khudanpur S. Recurrent neural network based language model. Annual Conference of the International Speech Communication Association. 2010. P. 1045–1048.

12. R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa Natural language processing (almost) from scratch. Journal of Machine Learning Research. 2011. Vol. 12. P. 2493–2537.

13. Rotim L., Snajder J. Comparison of Short-Text Sentiment Analysis Methods for Croatian. In Proceedings of the 6th Workshop on Balto-Slavic Natural Language Processing, Valencia, Spain, 4 April 2017. P. 699–731.

14. T. Young, D. Hazarika, S. Poria, E. Cambria Recent trends in deep learning based natural language processing. IEEE Comput. Intell. Mag. Aug. 2018. Vol. 13, No. 3. P. 55–75.

15. Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781. ICLR Workshop, 2013. P. 1-12.

16. Y. Kim. Convolutional neural networks for sentence classification. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, Association for Computational Linguistics, October 2014. P. 1746–1751.

17. S. Hochreiter and J. Schmidhuber. Long short-term memory. Neural Computation, 1997. Vol. 9. P. 1735–1780.

18. G. Katz, N. Ofek, and B. Shapira Context-based sentiment analysis, Knowledge-Based Systems. ConSent. 2015. Vol. 84, No. 1. P. 162–178.

19. Hogenboom, A., Bal, D., Frasincar, F., Bal, M., De Jong, F., & Kaymak, U. Exploiting emoticons in polarity classification of text. Journal of Web Engineering, 2015. Vol. 14. P. 22–40.

20. P. D. Turney Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews. Proceedings of the 40th annual meeting on association for computational linguistics. 2002. P. 417–42.

ДОДАТОК А ДЕМОНСТРАЦІЙНІ МАТРЕІАЛИ

Рисунок А.1

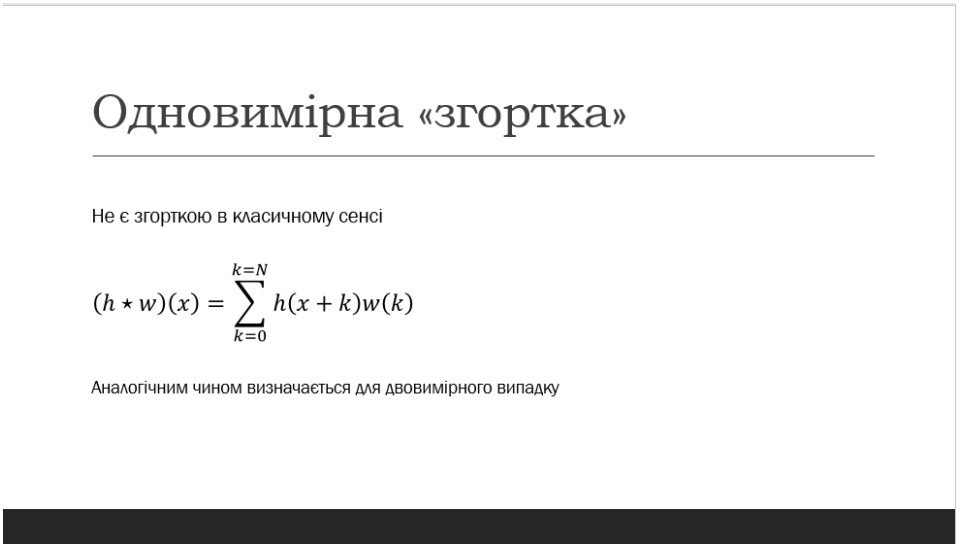

Рисунок А.2

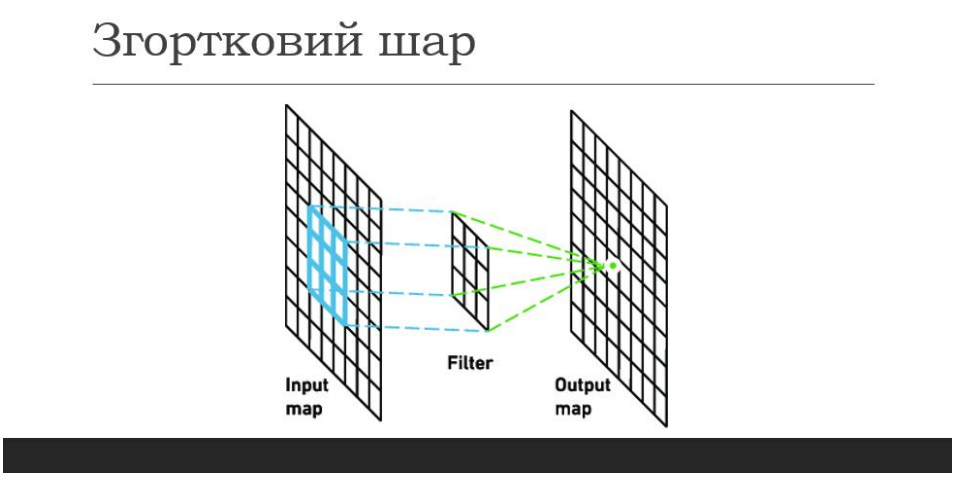

Рисунок А.3

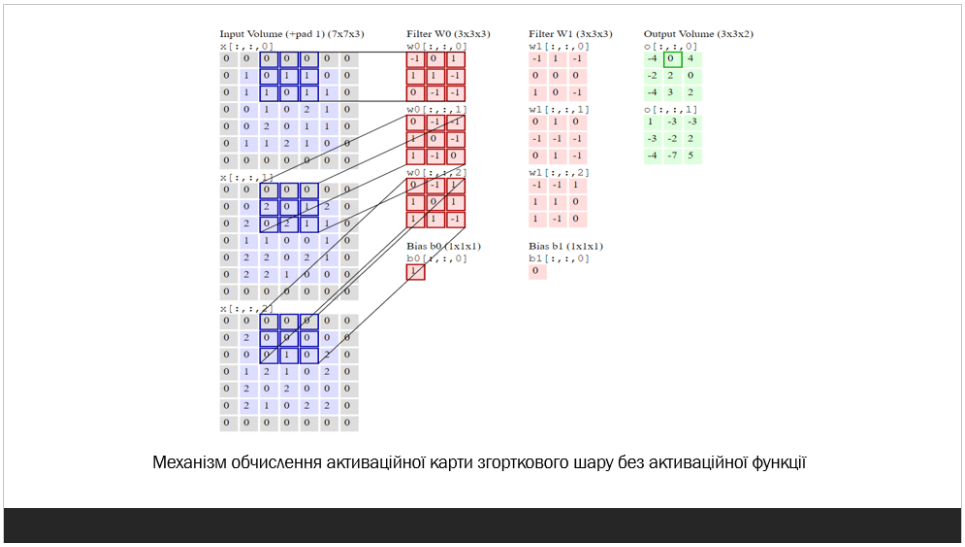

Рисунок А.4

Активаційні функції

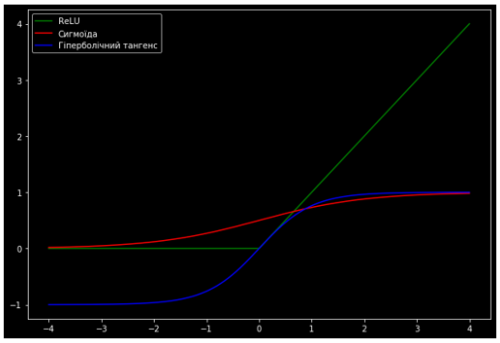

Рисунок А.5

VGG16

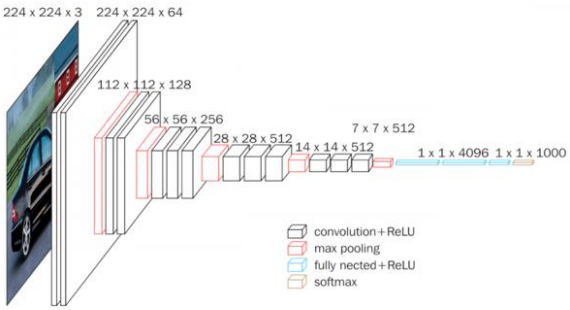

Рисунок А.6

Рисунок А.7

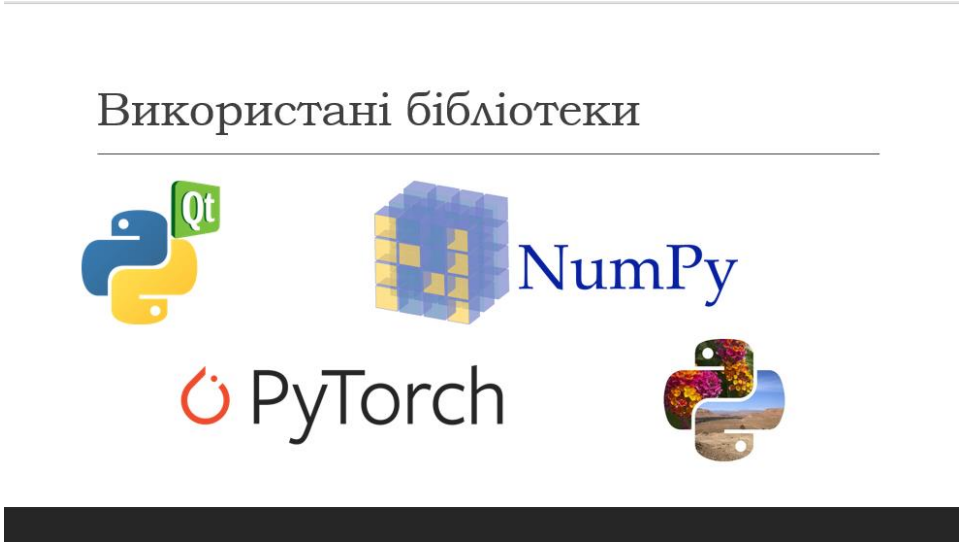

Рисунок А.8

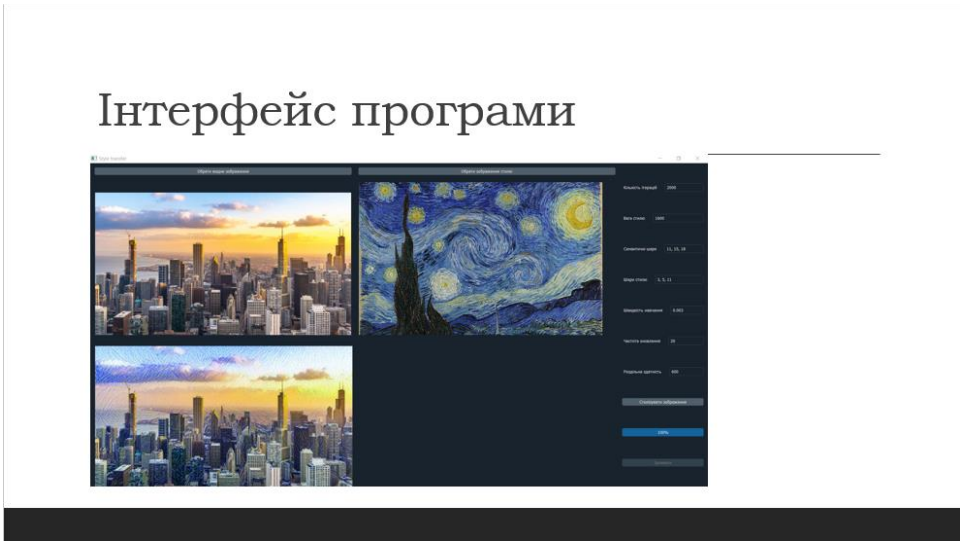

Рисунок А.9

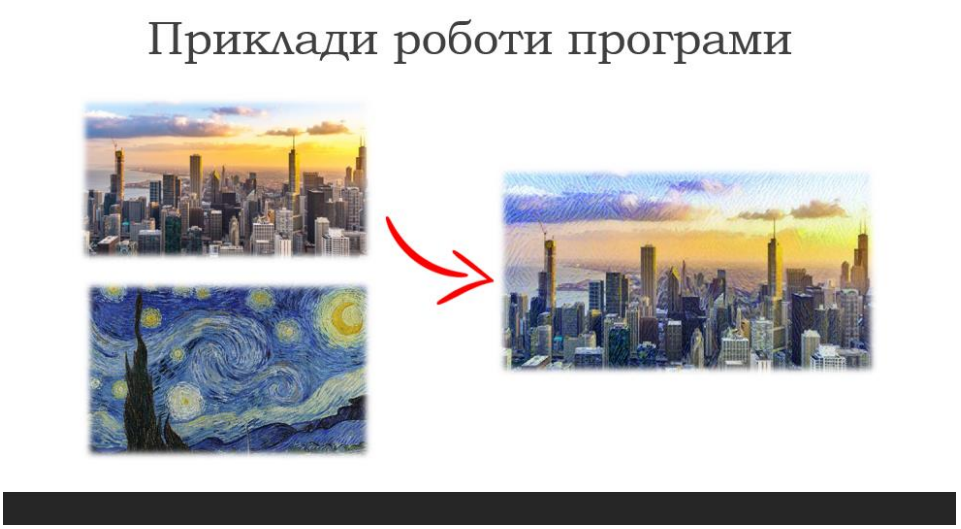

Рисунок А.10

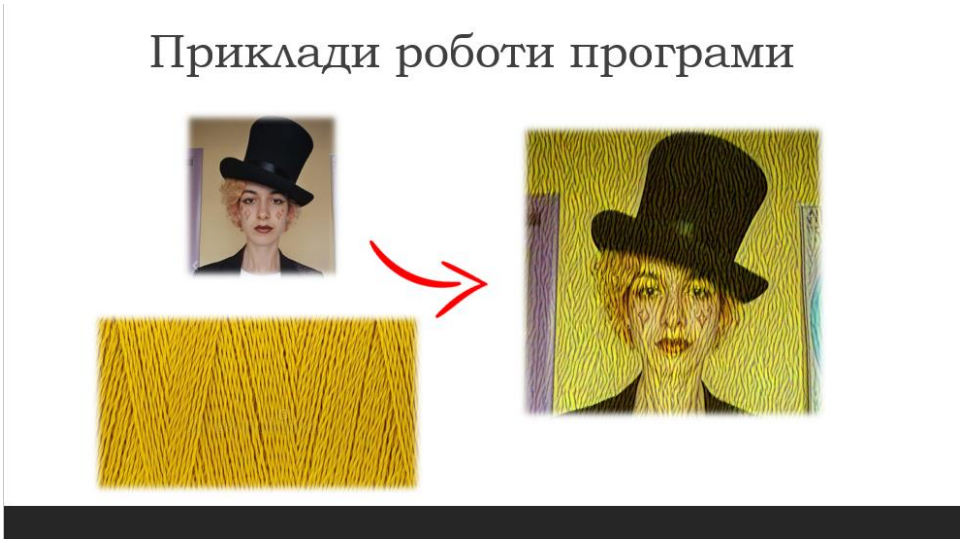

Рисунок А.11

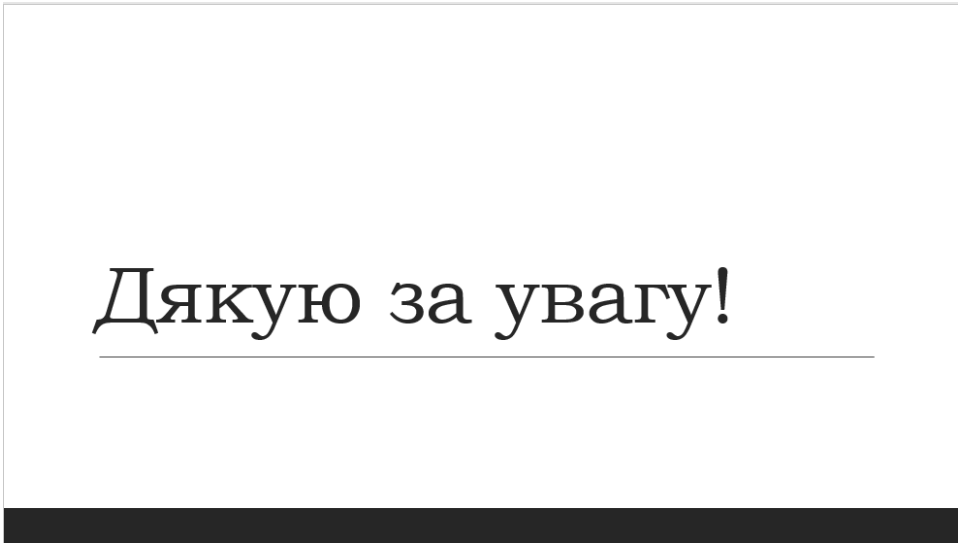

Рисунок А.12

ДОДАТОК В КОД ПРОГРАМНОГО ПРОДУКТУ

main.py

```
import os
import sys

from PIL import Image
import io

from PyQt5 import QtCore
from PyQt5.QtWidgets import QWidget, QGridLayout, QPushButton, QSpinBox, QTabWidget, QApplication, QLabel, \
    QFileDialog, QLineEdit, QProgressBar
from PyQt5.QtGui import QPixmap
from PyQt5.QtCore import Qt, pyqtSlot

import qdarkstyle

import matplotlib
matplotlib.use("Qt5Agg")
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas

from backend import Transferer

def connect(obj, func):
    if isinstance(obj, QSpinBox):
        obj.valueChanged.connect(func)
    else:
        obj.clicked.connect(func)
    return obj

class QTabWidgetFixed(QTabWidget):
    def showEvent(self, a0):
        try:
            return super().showEvent(a0)
        finally:
            if Qt.WindowState == Qt.WindowMaximized:
                self.setFixedSize(self.size())

class MainWindow(QWidget):
    def __init__(self, *args, **kwargs):
```

```
    super(MainWindow, self).__init__(*args, **kwargs)
    self.min_size = (800, 600)
    self.transferer = Transferer()
    self.source_image = None
    self.style_image = None
    self.target_image = None
    self.stop_flag = False
    self.__initUI__()

def __initUI__(self):

    self.choose_source_button = QPushButton('Обрати вхідне зображення')
    self.choose_source_button.clicked.connect(self.get_source_image)
    self.source_image_label = QLabel()
    self.source_image_label.setMinimumSize(*self.min_size)

    self.choose_style_button = QPushButton('Обрати зображення стилю')
    self.choose_style_button.clicked.connect(self.get_style_image)
    self.style_image_label = QLabel()
    self.style_image_label.setMinimumSize(*self.min_size)

    self.transfer_image_label = QLabel()
    self.transfer_image_label.setMinimumSize(*self.min_size)

    self.transfer_button = QPushButton('Стилізувати зображення')
    self.transfer_button.clicked.connect(self.get_transfer_image)

    self.stop_button = QPushButton('Зупинити')
    self.stop_button.clicked.connect(self.stop_transfer)
    self.stop_button.setDisabled(True)

    self.iterations_amount_edit = QLineEdit()
    self.iterations_amount_edit.setText('100')

    self.style_weight_edit = QLineEdit()
    self.style_weight_edit.setText('100')

    self.content_layers_edit = QLineEdit()
    self.content_layers_edit.setText('11, 15, 18')

    self.style_layers_edit = QLineEdit()
    self.style_layers_edit.setText('3, 5, 11')
```

```
self.resolution_edit = QLineEdit()
self.resolution_edit.setText('500')

self.display_frequency_edit = QLineEdit()
self.display_frequency_edit.setText('1')

self.lr_edit = QLineEdit()
self.lr_edit.setText('0.003')

self.progress_bar = QProgressBar()

self.progress_bar.setMaximum(self.iterations_amount)

main_layout = QGridLayout()
main_layout.setAlignment(Qt.AlignCenter)
main_layout.setVerticalSpacing(20)
main_layout.setHorizontalSpacing(20)
main_layout.addWidget(self.choose_source_button, 0, 0)
main_layout.addWidget(self.source_image_label, 1, 0)

main_layout.addWidget(self.choose_style_button, 0, 1)
main_layout.addWidget(self.style_image_label, 1, 1)

main_layout.addWidget(self.transfer_image_label, 2, 0, 1, 2)

right_tab_layout = QGridLayout()

sub_layout = QGridLayout()
sub_layout.addWidget(QLabel('Кількість ітерацій'), 0, 0)
sub_layout.addWidget(self.iterations_amount_edit, 0, 1)

right_tab_layout.addLayout(sub_layout, 0, 0)

sub_layout = QGridLayout()
sub_layout.addWidget(QLabel('Вага стилю'), 0, 0)
sub_layout.addWidget(self.style_weight_edit, 0, 1)

right_tab_layout.addLayout(sub_layout, 1, 0)

sub_layout = QGridLayout()
sub_layout.addWidget(QLabel('Семантичні шари'), 0, 0)
```

```
        sub_layout.addWidget(self.content_layers_edit, 0, 1)

        right_tab_layout.addLayout(sub_layout, 2, 0)

        sub_layout = QGridLayout()
        sub_layout.addWidget(QLabel('Шари стилю'), 0, 0)
        sub_layout.addWidget(self.style_layers_edit, 0, 1)

        right_tab_layout.addLayout(sub_layout, 3, 0)

        sub_layout = QGridLayout()
        sub_layout.addWidget(QLabel('Швидкість навчання'), 0, 0)
        sub_layout.addWidget(self.lr_edit, 0, 1)

        right_tab_layout.addLayout(sub_layout, 4, 0)

        sub_layout = QGridLayout()
        sub_layout.addWidget(QLabel('Частота оновлення'), 0, 0)
        sub_layout.addWidget(self.display_frequency_edit, 0, 1)

        right_tab_layout.addLayout(sub_layout, 5, 0)

        sub_layout = QGridLayout()
        sub_layout.addWidget(QLabel('Роздільна здатність'), 0, 0)
        sub_layout.addWidget(self.resolution_edit, 0, 1)

        right_tab_layout.addLayout(sub_layout, 6, 0)

        right_tab_layout.addWidget(self.transfer_button, 7, 0)
        right_tab_layout.addWidget(self.progress_bar, 8, 0)
        right_tab_layout.addWidget(self.stop_button, 9, 0)

        main_layout.addLayout(right_tab_layout, 0, 2, 3, 1)

        self.setLayout(main_layout)

    def closeEvent(self, event):
        self.stop_flag = True
        event.accept()

    @property
    def iterations_amount(self):
```

```
        return int(self.iterations_amount_edit.text())

    @property
    def style_weight(self):
        return float(self.style_weight_edit.text())

    @property
    def content_layers(self):
        return [int(layer.strip()) for layer in self.content_layers_edit.text().split(sep=',')]

    @property
    def style_layers(self):
        return [int(layer.strip()) for layer in self.style_layers_edit.text().split(sep=',')]

    @property
    def display_frequency(self):
        return int(self.display_frequency_edit.text())

    @property
    def lr(self):
        return float(self.lr_edit.text())

    @property
    def resolution(self):
        return int(self.resolution_edit.text())

    @staticmethod
    def _get_byte_image(image):
        byte_image = io.BytesIO()
        image.save(byte_image, format="JPEG")
        return byte_image.getvalue()

    def _set_image_to_label(self, image_label, image):
        source_map = QPixmap()
        source_map.loadFromData(self._get_byte_image(image))
        image_label.setPixmap(source_map.scaled(*self.min_size, QtCore.Qt.KeepAspectRatio))

    def _get_image_from_file(self):
        fn = QFileDialog.getOpenFileName(
            self, 'Open file', os.path.join(os.getcwd(), 'images'), "Image files (*.jpg)")[0]
        if not fn:
            return None
```

```
    image = Image.open(fn)
    return image

@pyqtSlot()
def get_source_image(self):
    image = self._get_image_from_file()
    if not image:
        return
    self.source_image = image
    self._set_image_to_label(self.source_image_label, self.source_image)

@pyqtSlot()
def get_style_image(self):
    image = self._get_image_from_file()
    if not image:
        return
    self.style_image = image
    self._set_image_to_label(self.style_image_label, self.style_image)

@pyqtSlot()
def get_transfer_image(self):
    self.stop_flag = False
    self.progress_bar.setMaximum(self.iterations_amount)
    self.stop_button.setEnabled(True)
    self.transfer_button.setDisabled(True)

    display_frequency = self.display_frequency

    self.transferer.init_training(
        self.source_image, self.style_image, self.style_weight, self.content_layers, self.style_layers, self.lr,
        self.resolution)

    for step in range(self.iterations_amount):
        if self.stop_flag:
            break

        try:
            self.transferer.train_step()
        except RuntimeError as e:
            print(e)
            break
```

```
            if not (step + 1) % display_frequency:
                self._set_image_to_label(self.transfer_image_label, self.transferer.get_image())

            self.progress_bar.setValue(step + 1)
            QApplication.processEvents()

        self.stop_button.setDisabled(True)
        self.transfer_button.setEnabled(True)

    @pyqtSlot()
    def stop_transfer(self):
        self.stop_flag = True


if __name__ == '__main__':
    app = QApplication(['Style transfer'])
    app.setStyleSheet(qdarkstyle.load_stylesheet_pyqt5())
    app.setApplicationName('Style transfer')
    main = MainWindow()
    main.setWindowTitle('Style transfer')
    main.showMaximized()
    sys.exit(app.exec_())
```

backend.py

```
from PIL import Image
import numpy as np
import torch
from torchvision import transforms

from pretrained_net import PretrainedNet


def process_image(image, transform=None, max_size=None, shape=None, device=None):
    if max_size:
        scale = max_size / max(image.size)
        size = np.array(image.size) * scale
        image = image.resize(size.astype(int), Image.ANTIALIAS)

    if shape:
```

```
        image = image.resize(shape, Image.LANCZOS)

    if transform:
        image = transform(image).unsqueeze(0)

    image = torch.tensor(image, requires_grad=False)

    if device:
        image = image.to(device)

    return image


class Transferer:
    def __init__(self):
        self.device = 'cuda:0'
        self.transform = transforms.Compose(
            [transforms.ToTensor(), transforms.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225))])

        self.net = None

        self.content_features = self.style_features = None
        self.style_weight = None
        self.max_size = None
        self.target = None
        self.optimizer = None
        self.step = None

        self.log_frequency = 10

    def process_source(self, source):
        return process_image(source, self.transform, max_size=self.max_size, device=self.device)

    def process_style(self, style, source):
        return process_image(style, self.transform, shape=[source.shape[2], source.shape[3]], device=self.device)

    def init_training(self, source, style, style_weight, content_layers, style_layers, lr, max_size):
        if self.net:
            del self.net
        self.net = PretrainedNet(content_layers, style_layers).to(self.device).eval()
        self.target = self.process_source(source)
        self.style_weight = style_weight
```

```
    self.max_size = max_size
    self.optimizer = torch.optim.Adam([self.target], lr=lr)
    self.step = 0

    with torch.no_grad():
        self.content_features = [tensor.data.clone() for tensor in self.net(self.target)[0]]
        self.style_features = [tensor.data.clone() for tensor in self.net(self.process_style(style, self.target))[1]]

    self.target.requires_grad = True

def style_loss(self, target_features):
    style_loss = 0
    content_loss = 0

    for f1c, f1s, f2, f3 in zip(target_features[0], target_features[1], self.content_features, self.style_features):
        content_loss += torch.mean((f1c - f2) ** 2)

        _, c, h, w = f1s.shape
        f1s = f1s.view(c, h * w)
        f3 = f3.view(c, h * w)

        target_gram = torch.mm(f1s, f1s.t())
        style_gram = torch.mm(f3, f3.t())

        style_loss += torch.sum((target_gram - style_gram) ** 2) / (4 * (c * h * w) ** 2)

    loss = content_loss + self.style_weight * style_loss
    losses = {
        'content': content_loss.item(),
        'style': style_loss.item()
    }
    return loss, losses

def train_step(self):
    target_features = self.net(self.target)
    loss, losses = self.style_loss(target_features)
    loss.backward()
    self.optimizer.step()
    self.optimizer.zero_grad()

    if not self.step % self.log_frequency:
        print(f'\nStep {self.step}')
```

```
            for name, value in losses.items():
                print(f'\t{name} loss: {value:.4f}')

        self.step += 1

    def get_image_tensor(self):
        denorm = transforms.Normalize((-2.12, -2.04, -1.80), (4.37, 4.46, 4.44))
        img = self.target.detach().cpu().data.squeeze()
        img = denorm(img).clamp_(0, 1)
        return img

    def get_image(self):
        return transforms.ToPILImage()(self.get_image_tensor())
```

pretrained_net.py

```
import torch.nn as nn
from torchvision import models


class PretrainedNet(nn.Module):
    def __init__(self, content, style):
        super(PretrainedNet, self).__init__()

        self.content = content
        self.style = style

        self.last = max(*content, *style)

        self.model = models.vgg16(pretrained=True).features.requires_grad_(False)

    def forward(self, x):
        features_content = []
        features_style = []

        for name, layer in self.model._modules.items():
            name = int(name)

            if name > self.last:
                return features_content, features_style
```

```
        x = layer(x)

        if name in self.content:
            features_content.append(x)

        if name in self.style:
            features_style.append(x)

    return features_content, features_style
```